GIPF

Zadanie składa się z dwóch części:

- Pierwszą jest implementacja silnika uogólnionej wersji gry. Uogólnienie wprowadza listę **L** parametrów opisujących grę. Pozwala to na łatwą zmianę charakterystycznych jej własności, np.: rozmiar planszy, liczba pionów wymaganych dla pewnej kombinacji wyzwalającej pewien efekt w grze, np. bicie/odzyskiwanie pionów, itp.
- Drugą jest implementacja programu rozwiązującego tę grę albo pewne jej stany (np. stany końcowe) dla różnych wartości parametrów listy **L**. Innymi słowy wariant gry opisany niskimi wartościami **L** będzie stanowił dużo mniejszy problem do całkowitego rozwiązanie gry, niż wariant opisany większymi wartościami **L** np. oryginalny. W tym drugim przypadku można próbować rozwiązywać stany gry gdzie "widać", że gra po kilku posunięciach powinna się zakończyć. Podstawowym algorytmem rozwiązującym będzie algorytm min-max (ew. jego wariacja negamax) wraz z przycinaniem alpha–beta. Na dodatkowe punkty można zaimplementować bardziej wyszukane usprawnienia, takie jak: pogłębianie iteracyjne, porządkowanie posunięć, z zastosowaniem Triangular PV-Table lub tablic transpozycji, itp. Alternatywnie można zaimplementować algorytm z rodziny best-first search, np. Proof Number Search w tym przypadku również można zdobyć dodatkowe punkty za implementacje jego bardziej wyrafinowanych wersji, np. PNS^2, Depth-first PN-Search (df-pn), Monte-Carlo PNS itp.

GIPF to gra deterministyczna, bez elementów losowych oraz ukrytej informacji dla dwóch graczy o sumie zerowej bez remisów. Zasady wersji oficjalnej dostępne są tutaj Zadanie polega na implementacji silnika tej gry dla jej uogólnionej wersji. Uogólnienie polega na tym, że można zmieniać właściwości gry opisane czteroma paramtrami (S, K, GW, GB) będącymi elementami wspomnianej wcześniej listy **L** parametrów opisujących grę:

- S - rozmiar planszy wyrażony liczbą pól planszy wchodzących w skład każdego boku sześciobocznej planszy;
- K - liczba pionów gracza która wyzwala zbieranie pionów, ta wartość nigdy nie powinna być mniejsza od dwóch ponieważ dla jedynki pozycja początkowa na planszy była by zabroniona (wszystkie piony należało by od razu usunąć). Wartość ta nie powinna być również większa od 2*S-1 ponieważ z tylu pól składa się "przekątna" planszy więc nie jest możliwe utworzenie dłuższego ciągu. Idąc dalej tym tropem można zauważyć, że wartość ta powinna być jeszcze mniejsza ponieważ w przypadku kiedy jest równa dokładnie 2*S-1 w grze nie można będzie zbijać pionków przeciwnika. Prowadzi to albo do niekończącej się rozgrywki polegającej na uwalnianiu w nieskończoność swoich pionów do rezerwy albo na zakończeniu rozgrywki poprzez zapchanie planszy, kiedy to żaden z graczy nie może dołożyć już żadnego piona do gry. Opisuje to jednoznacznie zasada z instrukcji: "Nie można wypchnąć piona poza obszar gry, czyli na kropkę po drugiej stronie linii.";
- GW - liczba pionów należących do gracza białego, musi być większa niż trzy które są na starcie umieszczane na planszy gracza. jednak w przypadku kiedy będzie ich mało rozgrywka może się szybko zakończyć z powodu wyczerpania zasobów gracza, wtedy gracz rozpoczynający, czyli biały będzie zawsze na przegranej pozycji (dotyczy to przypadku równej liczby pionów u obydwu graczy);
- GB - liczba pionów należących do gracza czarnego, może się różnić od GW.

Jak w podstawowej wersji każdy gracz zaczyna grę z 3 pionami na planszy, zawsze rozpoczyna biały. Piony umieszcza się na kropkach w rogach, a następnie przesuwa na pierwszy punkt w kierunku środka obszaru gry. Celem gracza jest, zdobycie pionów przeciwnika tak aby nie miał on żadnych pionów w rezerwie. Jednak jak zauważyliśmy wcześniej przy pewnych niefortunnych wariantach gry (niefortunnie dobrane parametry listy **L**) rozgrywka może się nie skończyć albo plansza może zostać zapchana. W pierwszym przypadku wynik rozwiązania gry będzie remisem, a w drugim gracz który nie może wykonać posunięcia będzie przegrywającym. jednak takie warianty gry będziemy traktować jako niewłaściwe starając się nie wykorzystywać ich w testach. Jak łatwo zauważyć, oryginalna gra opisana jest listą **L**=(4, 4, 15, 15) gdzie jedna z ostatnich dwóch wartości może być zwiększona maksymalnie o 3 w celu wprowadzenia asymetrii i tym samym wyrównaniu różnic w poziome gry obydwu graczy. Najmniejsza plansza jaką można sobie wyobrazić to plansza rozmiaru R = 2. Mniejsza plansza nie ma sensu ponieważ dla R = 1 obszar gry składał by się z tylko jednego pola gry, a musi ich być przynajmniej sześć aby pomieścić wszystkie początkowe (2x3) piony graczy.

Silnik gry powinien pozwalać na:

- Wczytanie stanu gry, będzie on podawany w przypadkach testowych w następującym formacie. W pierwszej linii będą podane cztery liczby opisujące grę (zawartość listy **L**). W kolejnej linii pojawi się liczba pionów w rezerwach odpowiednio gracza grającego białymi i czarnymi oraz cyfra określająca aktywnego gracza. W kolejnych liniach znajdą się informacje opisujące stan planszy.
  - Początkowa plansza oryginalnej wersji gry będzie opisana w następujący sposób:

```
4 4 15 15
```

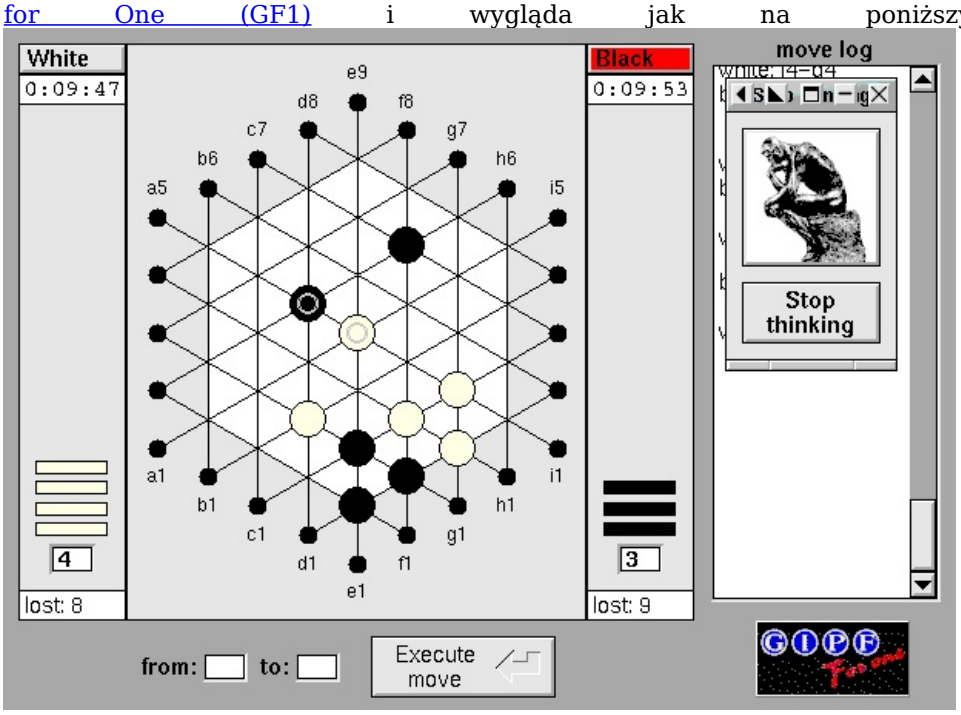
```
12 12 W
   W _ _ B
  _ _ _ _ _
 _ _ _ _ _ _
B _ _ _ _ _ W
 _ _ _ _ _ _
  _ _ _ _ _
   W _ _ B
```

- Mniejsze przykładowe początkowe plansze mogą być opisane w następujący sposób:

```
2 2 5 5
2 2 W
 W B
B _ W
 W B
```

```
3 3 10 9 W
7 6 W
  W _ B
 _ _ _ _
B _ _ _ W
 _ _ _ _
  W _ B
```

- Zastosowanie posunięcia dla aktualnej planszy opisanego wg schematu: xN - yM gdzie xN to pole na które zostaje położony pion z rezerwy gracza, a pole yM to pole w kierunku którego ten właśnie położony pion zostaje przesunięty. Notacja ta jest zapożyczona z programu Gipf for One (GF1) i wygląda jak na poniższym rysunku:



Zauważmy, że sąsiednie kropki nie będą stosowane w reprezentacji planszy, ponieważ używane są one jedynie podczas pierwszej fazy posunięcia, kiedy gracz kładzie piona z rezerw. Po położeniu piona na kropce gracz musi go przesunąć, jeśli takie przesunięcie spowoduje wypchnięcie piona po przeciwnej stronie na kropkę, to jest ono niedozwolone. W takim przypadku silnik gry, powinien zapamiętać to posunięcie jako błędne i nie powinien już analizować kolejnych posunięć. W przypadku prośby o wydrukowanie stanu planszy będzie drukowany jej stan bezpośrednio poprzedzający ten niedozwolony ruch. W przypadku gdy w tym samym czasie powstanie więcej rzędów składających się z 4 (bądź więcej) pionów tego samego koloru i rzędy te przecinają się, rozkaz musi zawierać informację który rząd ma być usunięty, w innym przypadku będzie to błędne posunięcie i konsekwencje będą identyczne jak w opisanych wcześniej przypadkach.
- Wydrukowanie stanu planszy w postaci identycznej jak przy jej wczytywaniu, tak aby możliwa była ewentualna rozgrywka pomiędzy dwoma programami.
- Wydrukowanie stanu gry:
  - in_progress - gra się nie skończyła
  - white_win - wygrał rozpoczynający
  - black_win - wygrał gracz drugi
  - dead_lock <kolor> - gracz którego tura właśnie trwa a on nie może wykonać posunięcia kolor gracza, dodatkowo drukujemy kolor gracza
  - bad_move <kolor> <xN - yM> - gracz wykonał błędne posuniecie, dodatkowo drukujemy kolor gracza i komendę która wygenerowała błędne posunięcie
- Generowanie wszystkich możliwych posunięć aktywnego gracza.
- Ocenę gry w postaci odpowiedzi na pytanie - czy gra się zakończyła i czy gracz wygrał/zremisował/przegrał? Zauważmy, że jest to równoważne z odpowiedzią na pytanie - czy aktywny gracz może wykonać jakiekolwiek posunięcie. jeśli nie to przegrał a jego przeciwnik

wygrał. Remis może nastąpić tylko w przypadku stwierdzenia, że gra nigdy się nie skończy, jednak na potrzeby tego zadania będziemy starali się nie doprowadzać do takich sytuacji.
- W zoptymalizowanej wersji, w przypadku kiedy przynajmniej jedno z posunięć aktywnego gracza prowadzi do jego wygranej, silnik powinien wygenerować tylko jedno z tych posunięć.

Solver, czyli algorytm rozwiązujący grę powinien pozwalać na:

- Rozwiązywanie gry na małej planszy, albo stanów końcowych na bardziej złożonej planszy.
- Odpowiedź na pytanie czy w zadanej liczbie posunięć dla zadanego stanu gry (posunięcie rozumiemy jako ruch pojedynczego pracza, a nie obydwu) aktywny gracz może wygrać.

Wymagania implementacyjne.

Program powinien reagować na następujące komendy:

- LOAD_GAME_BOARD po której podawane są parametry gry oraz stan planszy. Przykładowo, wczytanie oryginalnej planszy będzie wyglądało następująco:

```
LOAD_GAME_BOARD
4 4 15 15
12 12 W
   W _ _ B
  _ _ _ _ _
 _ _ _ _ _ _
B _ _ _ _ _ W
 _ _ _ _ _ _
  _ _ _ _ _
   W _ _ B
```

  Należy sprawdzić czy podany stan planszy jest poprawny, czy suma pionów obydwu graczy na planszy i w rezerwach jest poprawna, czyli identyczna z podaną w liście **L**. Należy sprawdzić czy nie zapomnieliśmy usunąć ciągów pionów, które spełnią kryterium wyzwalające ich zbieranie.
- PRINT_GAME_BOARD drukuje aktualny stan planszy, po wczytaniu gry powinien to być dokładnie ten sam wydruk który został przekazany podczas wczytywania. Oczywiście zakładając, że był on poprawny.
- DO_MOVE <xN - yM> wykonuje przekazany ruch dla aktywnego gracza na aktualnej planszy. Notacja posunięć powinna być zgodna z tą z programu Gipf for One (GF1) W przypadku gdy będzie on poprawny otrzymamy nową planszę a aktywny gracz zmieni się na przeciwnika. Może okazać się, że gra się zakończy jeśli przeciwnik nie będzie miał już pionów w rezerwie. W przypadku gdy ruch będzie niepoprawny należy ustalić odpowiedni stan gry, czyli "bad_move" oraz zapamiętać gracza i ruch które ten stan wywołały. W przypadku kiedy gracz musi zdjąć piony z planszy a istnieje kilka takich możliwości, musi on zadecydować które piony chce zdjąć. Ta informacja musi zostać dodana do komendy w postaci współrzędnych tych pionów. Współrzędne pionów są podawane wg schematu opisanego przykładem początkowej planszy dla standardowej wesji gry:

```
    + + + + +
   + W _ _ B +
  + _ _ _ _ _ +
 + _ _ _ _ _ _ +
+ B _ _ _ _ _ W +
 + _ _ _ _ _ _ +
  + _ _ _ _ _ +
   + W _ _ B +
    + + + + +
```

  Indeksy pól powyższej planszy:

```
    a5 b6 c7 d8 e9
   a4 b5 c6 d7 e8 f8
  a3 b4 c5 d6 e7 f7 g7
 a2 b3 c4 d5 e6 f6 g6 h6
a1 b2 c3 d4 e5 f5 g5 h5 i5
  b1 c2 d3 e4 f4 g4 h4 i4
    c1 d2 e3 f3 g3 h3 i3
      d1 e2 f2 g2 h2 i2
        e1 f1 g1 h1 i1
```

  Indeksy pól mniejszych plansz:

```
  a3 b4 c5
 a2 b3 c4 d4
a1 b2 c3 d3 e3
  b1 c2 d2 e2
    c1 d1 e1

   a4 b5 c6 d7
```

```
  a3 b4 c5 d6 e6
 a2 b3 c4 d5 e5 f5
a1 b2 c3 d4 e4 f4 g4
  b1 c2 d3 e3 f3 g3
    c1 d2 e2 f2 g2
      d1 e1 f1 g1
```

Zatem w przypadku ruchu który doprowadzi do niejednoznacznej sytuacji wymagającej dookreślenie które piony trzeba zdjąć. Po "normalnym" rozkazie podawany jest kolor gracza którego dotyczy wybór a nastepnie ciąg współrzędnych określających zdejmowane piony. Rozkaz będzie wyglądał nastepująco:
DO_MOVE <xN - yM> [w:|b:] x1 xn
W przypadku kiedy tych wyborów będzie więcej zostana one podane w postaci kolejnych ciągów indeksów.

- PRINT_GAME_STATE drukuje aktualny stan gry ("in_progress", "bad_move", itp.).
- GEN_ALL_POS_MOV drukuje listę wszystkich posunięć (bez powtórzeń) prowadzących do unikalnych stanów planszy.
- GEN_ALL_POS_MOV_EXT działa identycznie jak GEN_ALL_POS_MOV z wyjątkiem przypadku kiedy istnieje dowolne posunięcie wygrywające, wtedy drukuje tylko jedno z tych posunięć.
- GEN_ALL_POS_MOV_NUM drukuje liczbę wszystkich posunięć jakie otrzymujemy w wyniku działania komendy GEN_ALL_POS_MOV.
- GEN_ALL_POS_MOV_EXT_NUM działa identycznie jak GEN_ALL_POS_MOV_NUM z wyjątkiem przypadku kiedy istnieje dowolne posunięcie wygrywające, wtedy drukuje 1.
- WINNING_SEQUENCE_EXIST <N> odpowiada na pytanie czy istnieje dla któregokolwiek z graczy wygrywająca sekwencja posunięć mniejsza równa N.
- SOLVE_GAME_STATE odpowiedź na pytanie dla którego z graczy dany planszy gry jest wygrywający. Możliwe są dwie odpowiedzi WHITE_HAS_WINING_STRATEGY albo BLACK_HAS_WINING_STRATEGY.

Kryterium oceny

Zadanie nie będzie w pełni automatycznie testowane, a student będzie musiał zaprezentować które funkcjonalności udało się zaimplementować. Proszę zatem na dołączenie w postaci pliku tekstowego wszystkich komend na które program powinien reagować. Przykładowo mile widziane będą zestawy komend polegające na wczytaniu pewnego stanu gry, wykonaniu pewnej liczby posunięć, wydrukowaniu planszy oraz stanu gry, a następnie sprawdzeniu czy jest on zgodny z oczekiwaniami. powyższe scenariusze powinny dotyczyć serii posunięć prowadzących do wszystkich możliwych stanów gry (in_progress, itp.). Podczas prezentacji funkcjonalności silnika gry generującej listę wszystkich możliwych posunięć, dodatkowo należy zaprezentować, usprawnienie polegające na wygenerowaniu tylko jednego posunięcia wygrywającego (oczywiście jeśli takowe istnieje). Wagi oceny prezentują się następująco:

- Poprawne wczytanie planszy. (10%) Test 0
- Poprawne wydrukowanie planszy. (10%) Test 1
- Poprawne wykonanie posunięcia nie zmieniającego stanu gry, chodzi o "zwykłe posunięcia" w stanie gry "in_progress". Test 2-5 (40%)
- Prezentacja posunięć prowadzących do wszystkich możliwych stanów gry stanu gry, czyli powodujące przejście ze stanu gry "in_progress" w "white_win", black_win, itp. (10%)
- Wygenerowanie wszystkich możliwych posunięć dla wczytanego stanu gry. (10%) Test 6
- Detekcja posunięcia wygrywającego podczas generowania wszystkich możliwych posunięć, widoczna poprzez wydrukowanie pojedynczego posunięcia wygrywającego. (10%) Test 7
- Prezentacja poprawnego działania Solver'a odpowiadającego czy dany gracz może wygrać/przegrać w zadanej, liczbie posunięć. Student powinien wygenerować takie stany gry, na których do zakończenia gry pozostało N posunięć gdzie $N \geq 1$ i $N \leq 2$. (10%)
- To samo co wyżej ale dla $N > 2$ i większych. Wiadomo, że dla mniejszych plansz powinno to być to łatwiejsze. (10%)
- Prezentacja prostych, końcowych stanów gry, które udało się rozwiązać. (10%) Test 9
- Prezentacja bardziej ambitnych stanów gry, które udało się rozwiązać wraz z sekwencją posunięć wygrywających w postaci drzewa dowodzącego gdzie na jedno optymalne posunięcie gracza rozpoczynającego przypadają wszystkie możliwe posunięcia przeciwnika. (10%)
  Prezentacja wyników działania Solvera wykorzystującego bardziej zaawansowanie algorytmy, porównanie ze zwykłym mini-max'em + aplha-beta. Prezentacja faktu, że udało się rozwiązać bardziej złożone plansze albo zysku czasowego/pamięciowego . (10%-30% bonusowe)

Teoretycznie można uzyskać 160% czyli 56 pkt.

Dodatkowa literatura:

- GAME-TREE SEARCH USING PROOF NUMBERS: THE FIRST TWENTY YEARS

Testy

Tutaj

Test 0 - Wejście

Testowanie poprawności wczytywanej planszy. W pierwszej linii pojawi się instrukcja LOAD_GAME_BOARD a następnie plansza którą należy wczytać i sprawdzić jej poprawność pod kątem, liczby pionów oraz jej rozmiarów (liczby pól we wczytywanych liniach wejścia).

Test 0 - Wyjście

Jeden z poniższych stringów:

- BOARD_STATE_OK - plansza poprawna
- WRONG_WHITE_PAWNS_NUMBER - zła liczba białych pionów
- WRONG_BLACK_PAWNS_NUMBER - zła liczba czarnych pionów
- WRONG_BOARD_ROW_LENGTH - zła długość wiersza planszy

Test 1 - Wejście

Drukowanie planszy. W pierwszej linii pojawią się dodatkowa instrukcja: PRINT_GAME_BOARD drukująca zawartość planszy.

Test 1 - Wyjście

W odpowiedzi na PRINT_GAME_BOARD drukowana jest zawartość planszy o identycznym układzie jak ten z instrukcji LOAD_GAME_BOARD. W przypadku drukowania pustej planszy, co ma miejsce po próbie wczytania błędnej planszy pojawia się komunikat: "EMPTY_BOARD".

Test 2 - Wejście

Testowanie poprawności wykonywanych posunięć. W pierwszej linii pojawią się dodatkowe instrukcje:

- DO_MOVE [w:|b:] y1 yn - wykonuje posunięcie z pola x1, które musi być polem startowym w kierunku pola x2 które musi z nim sąsiadować.

Test 2 - Wyjście

Jeden z poniższych stringów:

- MOVE_COMMITTED - ruch poprawny, zatwierdzono zmiany na planszy i zmieniono aktualnego gracza.
- BAD_MOVE_<x1>_IS_WRONG_INDEX - zły indeks, pokazuje na pole spoza planszy. W przypadku kiedy podano więcej niż jeden zły indeks, drukowany jest tylko pierwszy napotkany.
- UNKNOWN_MOVE_DIRECTION - nie można określić kierunku ruchu.
- BAD_MOVE_<x1>_IS_WRONG_STARTING_FIELD - wybrano złe, pole startowe (powinno to być jedno z położonych na obrzeżach planszy)
- BAD_MOVE_<x2>_IS_WRONG_DESTINATION_FIELD - wybrano złe, pole docelowe (powinno to być pole wolne albo zawierające pion któregoś z graczy)
- BAD_MOVE_ROW_IS_FULL - nie można wykonać ruchu bo wiersz jest zapełniony.

Test 3 - Wejście

Testowanie poprawności wczytywanej planszy. W pierwszej linii pojawi się instrukcja LOAD_GAME_BOARD a następnie plansza którą należy wczytać i sprawdzić jej poprawność pod kątem istnienia wierszy zawierających ciąg długości przynajmniej K.

Test 3 - Wyjście

Jeden z poniższych stringów:

- BOARD_STATE_OK - plansza poprawna
- ERROR_FOUND_<N>_ROW_OF_LENGTH_K - błędna plansza znaleziono N wierszy zawierających ciąg długości przynajmniej K.

Test 4 - Wejście

Rozszerzenie testu 2 pozwala na testowanie poprawności wykonywanych posunięć i drukowania planszy w przypadku konieczności zbierania pionów. Piony zbierane są z całego ciągu w którym jeden kolor tworzy nieprzerwany podciąg długości K albo większej. Piony tego koloru wracają do rezerw gracza pozostałe piony nie wracają do gry.

Test 5 - Wejście

Rozszerzenie testu 5 pozwala na testowanie poprawności wykonywanych posunięć i drukowania planszy w przypadku konieczności zbierania pionów w niejednoznacznych sytuacjach w których należy doprecyzować który rząd ma zostać zdjęty.

- DO_MOVE <x1-x2> [w:|b:] y1 yn - w wyniku wykonania posunięcia należy usunąć ciąg białych albo czarnych pionów (z zależności czy podano w: czy b:) o skrajnych pionach na polach y1 i yn.

Uwaga w testach zarówno kolor gracza jak i skrajne piony mogą być podane błędnie w takim przypadku należy wydrukować adekwatną informację:

- WRONG_COLOR_OF_CHOSEN_ROW - podano błędny kolor.
- WRONG_INDEX_OF_CHOSEN_ROW - podano błędny index.

Test 6 - Wejście

Testowanie funkcjonalności generującej wszystkie możliwe ruchy gracza prowadzące do unikalnych stanów planszy. Na wejściu najpierw nastąpi wczytanie planszy komendą LOAD_GAME_BOARD. W kolejnym kroku nastąpi komenda GEN_ALL_POS_MOV_NUM. W tej komendzie nie są testowane posunięcia prowadzące do niejednoznacznego zbierania pionów z planszy.

Test 6 - Wyjście

Komenda <N>_UNIQUE_MOVES - zwracająca informację o liczbie N unikalnych stanów planszy.

Test 6a - Wejście

Test 6 z dodanymi komendami GEN_ALL_POS_MOV do komend GEN_ALL_POS_MOV_NUM.

Test 6b - Wyjście

Wydrukowane wszystkie możliwe stany gry, ten test nie jest umieszczany na STOS'ie aby nie wymuszać kolejności generowania stanów gry. Programy studentów powinny zachowywać się identycznie co do kolejności drukowanych stanów.

Testy 7 i 7a

Rozszerzenie testu 6 o funkcjonalność komendy GEN_ALL_POS_MOV_NUM generującej posunięcia prowadzące do niejednoznacznego zbierania pionów z planszy. Test 7a podobnie jak 6a nie jest wykonywany na STOS'ie a ma za zadanie dostarczyć szczegółowych informacji o wygenerowanych ruchach w przypadku problemów z testem 6. Testowanie metody IS_GAME_OVER.

Testy 8 i 8a

Testowanie metody GEN_ALL_POS_MOV_EXT_NUM.

Testy 9 i 9a

Testowanie metody SOLVE_GAME_STATE.