Piotr Kowalczyk

Dokumentacja projektu z Kursu Języka Lua

Temat: gra Reversi

Wrocław, 28 czerwca 2018

Spis treści

1. Wstęp

Reversi mimo upływu lat pozostaje klasyką gatunku. Proste zasady prowadzą do emocjonującej i rozwijającej rozgrywki, która nie pozwala w trywialny sposób określić strategii wygrywającej. To skłania kolejnych graczy, teoretyków gier, matematyków i informatyków do zmierzenia się z Reversi.

Założenia dotyczące mojego projektu cały czas ewoluowały. Pierwszym pomysłem było napisanie mocnych SI do Reversi i Dżungli, następnie pomyślałem, że o wiele bardziej sensownym doświadczeniem programistycznym w takim języku jak Lua jest stworzenie gry – zatem przesunąłem środek ciężkości projektu w stronę użycia frameworku Love 2D. Ostatecznie zrezygnowałem z tworzenia Dżungli i zająłem się tylko Reversi. Jednak główna idea przyświecająca projektowi (w zasadzie to całemu temu semestrowi) brzmiała: należy zdać egzamin z AiSDu w pierwszym terminie – i projekt przez cały czas tworzenia był zgodny z tym założeniem.

W dalszych rozdziałach zakładam, że zasady Reversi są znane lub poprawność ich implementacji nie podlega wątpliwości.

2. Opis użycia programu

Aby uruchomić grę, należy uruchomić program `run.sh`. Ukaże się okno gry. Większość okna zajmuje plansza do gry (ze startowym ustawieniem pionków). Po prawej stronie planszy znajduje się panel wyboru poziomu trudności. Aktualny poziom trudności jest podkreślony – aby go zmienić, należy kliknąć pożądany poziom trudności. Poniżej planszy, po lewej stronie znajduje się napis wskazujący na kolor pionków użytkownika. Kliknięcie na ten napis zmienia kolor pionków. Nieco na prawo jest napis wskazujący na kolor pionków, które poruszają się w tej turze. W prawym dolnym rogu znajduje się napis restart. Kliknięcie tego napisu spowoduje rozpoczęcie gry od początku bez zmiany ustawień. Zmiany ustawień (czyli poziomy trudności lub koloru pionków) także powoduje rozpoczęcie gry od nowa.

3. SI przygotowane do gry

Poniżej krótko opisane zostały boty (w kolejności od najsłabszego do najsilniejszego). Nazwy mają znaczenie historyczne (którego szukać można w tegorocznej edycji przedmiotu Sztuczna Inteligencja).

rand Gracz losowy (losuje spośród wszystkich możliwych ruchów z prawdopodobieństwem $\frac{1}{m}$, gdzie m = liczba ruchów).

idiot Ze wszyskich możliwych ruchów wybiera ten, po wykonaniu którego balans pionków będzie dla niego najlepszy.

novice Jak powyższy, z tym że porównuje najgorszy dla niego balans po wykonaniu dwóch ruchów (formalnie półruchów), zatem jest to MinMax z głębokością 2.

basic Też jest MinMaxem z głębokością 2, ale jego heurystyka zależy, oprócz balansu pionków, od balansu pionków w rogach planszy.

standard MinMax z głębokością 3, heurystyka jak wyżej.

flatmc Ten gracz implementuje algorytm Flat Monte Carlo. Rozgrywa 400 losowych gier, liczy zwycięstwa i na tej podstawie wybiera najlepsze posunięcie.

proto MinMax z głębokością 5, heurystyka jak w **basic**.

Uwaga: **flatmc** oraz **proto** są zdecydowanie wolniejsze od pozostałych algorytmów.

4. Interfejs SI

Program pozwala na pojedynkowanie się z własnoręcznie napisanym botem (oczywiście w języku Lua). Bot musi być poprawnie napisanym modułem Lua i wystawić funkcję `player(board, symbol)`, która jako argumenty bierze planszę (wraz z ustawieniem pionków) i symbol, którym oznaczone są pionki bota na planszy (gdzie `o` oznacza białe, a `#` – czarne). Aby zagrać z własnym (bądź cudzym) botem należy wpisać jego nazwę do tablicy `cpu` w pliku main.lua **zamiast** jednej z istniejących tam nazw. Dopisanie nowej nazwy na końcu tablicy nie umożliwi grania z nowym botem.

Można też przetestować swojego bota z użyciem skryptu rev.lua (np. w celu określenia poziomu trudności). Aby to zrobić, należy wpisać go jako najnowszą wartość zmiennej `prog1` bądź `prog2`, a pod drugą zmienną należy wpisać przeciwnika (jedną z zaprogramowanych wcześniej SI). Aby uruchomić skrypt należy wpisać w terminalu komendę `lua5.3 rev.lua` lub `luajit rev.lua`. Program automatycznie rozegra 1000 partii i w przystępny sposób przedstawi wyniki. Można podejrzeć ustawienia końcowe poszczególnych partii (odkomentowując opcję `verbose`) lub cały przebieg partii (odkomentowując opcję `my_debug`). Należy wziąć pod uwagę, że jeśli SI nie działa szybko, to wykonanie 1000 partii może długo trwać. Wtedy zaleca się wypić herbatę, a w ostateczności zmniejszyć ilość partii (czyli ustawić mniejszą wartość zmiennej `n`).

5. Dokumentacja funkcji

W tym rozdziale opisane zostały funkcje wykorzystane w implementacji (z pominięciem wyżej opisanych botów).

Plik rev_lib.lua:

`board_tostr(board)` Funkcja, która przepisuje argument (planszę) do czytelnego formatu string, używana do debugowania.

`make_free_board()` Funkcja, która zwraca pustą planszę.

`make_board()` Zwraca planszę z początkowym rozstawieniem pionków.

`deep_copy(board)` Zwraca kopię planszy. Używana przez niektóre SI.

`attacking(board, x, y, sym)` Zwraca `true`, gdy położenie pionka o symbolu `sym` na polu o współrzędnych `x, y` na planszy `board` jest poprawnym ruchem; w przeciwnym przypadku zwraca `false`. Nie sprawdza granic planszy.

`swap_checks(board, x, y, sym)` Odwraca na planszy `board` pionki, które należy obrócić po dołożeniu `sym` na polu `x, y`.

`move_good(board, x, y, sym)` Zwraca `true` gdy ruch jest poprawny: sprawdza granice planszy i wywołuje funkcję `attacking` z odpowiednimi argumentami. Gdy ruch niepoprawny, zwraca `nil`.

`poss_move(board, sym)` Zwraca `true`, gdy możliwy jest jakikolwiek poprawny ruch; w p. p. zwraca `false`. Wywołuje funkcję `attacking`.

`white_win(board)` Zwraca `true`, gdy białe mają więcej pionków niż czarne; `nil`, gdy tyle samo; `false`, gdy czarne mają więcej. textbfNie określa, czy rozgrywka się zakończyła.

`count(board, sym)` Zwraca dwie wartości: liczbę pionków gracza `sym` oraz liczbę pionków przeciwnika.

Plik rev.lua:

`play(p1, name1, p2, name2)` Rozgrywa partię między `p1` i `p2` (które są funkcjami `player` wystawionymi przez moduły SI) o nazwach, odpowiednio, `name1` i `name2` oraz w zrozumiały sposób przedstawia jej wynik. Gracz `p1` rozpoczyna partię. Pozwala testować swojego bota.

`test(p1, p2, n)` Rozgrywa `n` partii pomiędzy graczami `p1` i `p2` (którzy są funkcjami `player` wystawionymi przez moduły SI) oraz w estetyczny sposób przedstawia wyniki. Gracze rozpoczynają partie na zmianę. Służy do testowania botów.

Plik main.lua:

`restart()` Resetuje stan planszy; służy do rozpoczęcia nowej gry.

`love.load()` Ustawia rozmiar i tytuł okna oraz wywołuje `restart`.

`love.mousereleased(x, y, button, istouch)` Zbiera dane dotyczące kliknięcia lewym przyciskiem myszy i – w zależności od argumentów `x, y` – ustawia pionek, zmienia poziom trudności, zmienia kolory pionków lub restartuje rozgrywkę. Argument `istouch` jest pomijany.

`love.update(dt)` W zależności od tury wywołuje bota bądź umożliwa ruch graczowi. Argument `dt` jest pomijany.

`love.draw()` Rysuje planszę, pionki, poziom trudności i napisy.