Technologia Programowania 2017/2018 – Lista 4 (lab)

Termin: przed 2 stycznia, godz. 23:59

(Uwaga: przy wyliczaniu liczby spóźnień liczy się data umieszczenia oddawanej wersji programu w niezależnym repozytorium, np. github, riouxsvn)

Trylma (ang. chinese checkers)

Twoim celem jest zaprojektowanie i implementacja systemu do prowadzenia rozgrywek w chińskie warcaby. System należy tworzyć **w parach wykorzystując repozytorium**. Historia zmian w repozytorium będzie podstawą do oceny wkładu członków zespołu. Parowanie powinno się odbywać w ramach grup laboratoryjnych jednego prowadzącego. Jeśli nie masz partnera zgłoś się do prowadzącego laboratorium.

Zasady gry

Zasady są dosyć łatwe, więc poniżej przytoczę je jedynie pobieżnie. Przeczytaj opis zasad na wiki i tę instrukcję. W razie wątpliwości zapytaj mnie lub prowadzącego laboratorium.

1. W jednej grze uczestniczy 2, 3, 4 lub 6 graczy. Każdy gracz ma 10 pionów.
2. Gra toczy się na planszy w kształcie sześciopromiennej gwiazdy. Każdy promień zawiera 10 pól, w których początkowo umieszczane są piony. Wewnętrzny sześciokąt ma 61 pól, na każdy jego bok składa się 5 pól.
3. Celem każdego gracza jest umieszczenie wszystkich swoich pionów w przeciwległym promieniu. Wygrywa gracz, który dokona tego jako pierwszy. Pozostali gracze mogą kontynuować grę walcząc o kolejne miejsca.
4. Gracz rozpoczynający grę wybierany jest losowo. Następnie gracze wykonują ruchy po kolei zgodnie z kierunkiem ruchu wskazówek zegara.
5. Ruch polega na przesunięciu piona na sąsiednie puste pole lub na przeskakiwaniu pionem innych pionów (swoich lub przeciwnika) tak jak w warcabach, ale bez możliwości zbijania pionów. Ruch może odbywać się w dowolnym kierunku, poza przypadkiem, gdy pion wejdzie do docelowego (przeciwległego) promienia. Wówczas pion może wykonywać ruchy jedynie w obrębie tego promienia.
6. Dopuszczalne jest pozostawianie swoich pionów w promieniu docelowym innego gracza (tzw. blokowanie).
7. Gracz może zrezygnować z ruchu w danej turze.

Wymagania dotyczące systemu

1. System powinien działać w oparciu o architekturę klient-serwer. Zakładam, że na Kursie Programowania nauczyliście się używać gniazd i wątków. W razie wątpliwości warto przyjrzeć się przykładom z tej strony.
2. Gracz za pomocą aplikacji klienckiej powinien móc połączyć się z serwerem, dołączyć do gry i przeprowadzić rozgrywkę.
3. Serwer powinien weryfikować poprawność ruchów i pośredniczyć w komunikacji.
4. Do gry mogą zostać dołączeni gracze-boty działający na serwerze. Stwórz jedną przykładową implementację bota, która będzie wykonywała choć w miarę sensowne ruchy.

Dodatkowe wymagania

1. Postaraj się tworzyć system starannie i w przemyślany sposób. Wykorzystaj UML by zaproponować i rozwijać projekt systemu. Systemy chaotyczne zaprojektowane i pisane na ostatnią chwilę będą nisko ocenione. Bierz również pod uwagę, że to co stworzysz będzie miało wpływ na kolejną listę.

2. Stosuj podejście iteracyjne i przyrostowe. Na każdych zajęciach począwszy od 4 grudnia, aż do ostatecznego oddania programu powinien być obecny przynajmniej jeden przedstawiciel pary i zaprezentować aktualną (działającą!) wersję systemu.

3. Istnieje bardzo wiele wariantów gry, np. inne mogą być rozmiary planszy, inna liczba pionów, inne zasady dotyczące możliwych ruchów i blokowania. **Projektuj system tak by można go było łatwo rozszerzyć o nowe warianty gry.**

4. Postaraj się wykorzystać poznane narzędzia i wzorce. Im więcej ich zastosujesz, tym wyżej będzie ocenione Twoje rozwiązanie. Ich użycie powinno być jednak zawsze uzasadnione.

5. Jeśli zdążysz, postaraj się umożliwić na Twoim serwerze obsługę wielu toczących się równolegle gier.

Punktacja

Za implementację całej funkcjonalności każdy student może otrzymać maksymalnie **100 punktów**. Poprawność kodu powinna być na bieżąco weryfikowana testami jednostkowymi napisanymi przy użyciu JUnit (**40 punktów**). Stopień pokrycia testami może zostać zweryfikowany przez prowadzącego np. za pomocą EclEmma. Na bieżąco wysyłaj dobrze opisane zmiany do repozytorium (**20 punktów**). Przy oddawaniu projektu powinieneś opisać jego strukturę i działanie istotnych elementów − stwórz w tym celu diagram klas i inne diagramy UML, które mogą być pomocne (**20 punktów**). Łącznie każdy student może otrzymać **180 punktów**.

Za każdy błąd w działaniu programu, który nie będzie wykryty w testach, a zostanie wykryty przez prowadzącego można stracić **10 punktów** (za listę można otrzymać ujemną liczbę punktów). Począwszy od 4 grudnia nieobecność na zajęciach przedstawiciela zespołu każdorazowo wiąże się z odjęciem **10 punktów** od wyniku końcowego każdego z członków zespołu. Ostateczne oddanie projektu wymaga obecności obu członków zespołu.

Powodzenia, J.L.