Technologie programowania - 2025
Lista 5

Gra w go

Należy zaprojektować i zaimplementować system do przeprowadzania rozgrywek w grę w go. Podstawowe zasady gry i wymagania dotyczące systemu są przedstawione poniżej. Zasady te są sformułowane w opraciu o tutorial.

Zadanie należy rozwiązać w parach wykorzystując repozytorium. Historia zmian w repozytorium będzie podstawą do oceny wkładu członków zespołu.

Uwaga: Robimy iterację 2 o funkcjonalności podanej poniżej. Program powinien więc posiadać funkcjonalności wersji 1 oraz 2. Wymagany jest osobny link na github dla każdej iteracji.

Zasady gry

Zasady są dosyć łatwe, więc poniżej przytoczę je jedynie pobieżnie. Opis bardziej szczegółowy znajduje się na wiki oraz tutaj.

1. W jednej grze uczestniczy dwóch graczy. Gra toczy się na planszy będącej siatką 19 poziomych i 19 pionowych linii tworzących 361 przecięć. Czasami gra się także na planszach 13 na 13 lub 9 na 9.

2. Gracze kładą na przemian czarne i białe kamienie na przecięciu linii. Rozpoczynają czarne. Plansza jest początkowo pusta. Celem gry jest otoczenie własnymi kamieniami obszaru większego niż obszar przeciwnika.

3. Kamieni raz postawionych na planszy nie można zabrać ani przesunąć, mogą natomiast zostać uduszone przez przeciwnika, jeśli stracą wszystkie oddechy. Po zabraniu ostatniego oddechu przeciwnik zabiera z planszy i przechowuje uduszone przez siebie kamienie (zwane jeńcami). Oddechem kamienia nazywamy niezajęte sąsiednie przecięcie (połączone linią z kamieniem). Na przykład kamień stojący samotnie na środku planszy ma cztery oddechy, a w rogu planszy - dwa.

4. Kamienie jednego koloru stojące obok siebie i połączone liniami tworzą łańcuch, który ma wspólne oddechy – można je zbić albo wszystkie razem albo żadnego (zobacz np. tutaj).

5. Gracz nie może pozbawić swojej grupy kamieni ostatniego oddechu, ani położyć kamienia w punkt bez oddechu. Wyjątkiem od reguły jest sytuacja, w której taki ruch dusi kamienie przeciwnika (zobacz np. tutaj).

6. Kształt, w którym gracze mogą naprzemiennie dusić kamień przeciwnika, zwany jest ko. W celu uniknięcia nieskończonych cykli gracz, którego kamień został uduszony w **ko**, nie może udusić kamienia przeciwnika w następnym ruchu (zobacz np. tutaj).

7. O kamieniach mówimy, że są żywe, jeżeli nie mogą być zbite przez przeciwnika. O kamieniach, które nie są żywe, mówimy, że są martwe. Puste punkty otoczone żywymi kamieniami tylko jednego gracza zwane są punktami wewnętrznymi. Wszystkie inne puste przecięcia są nazywane punktami niczyimi. Kamienie, które są żywe, ale stykają się z punktami niczyimi, są w **seki**. Punkty wewnętrzne otoczone przez żywe kamienie nie będące w seki są zwane **terytorium** (zobacz np. lekcje 5-9 tutaj).

8. Gracz zawsze może zrezygnować z ruchu. Gdy obaj gracze bezpośrednio po sobie zrezygnują z ruchu, gra się zatrzymuje. Następnie obaj gracze uzgadniają, które grupy kamieni są żywe, a które martwe, a także jakie są terytoria. Jeżeli gracze nie mogą dojść do porozumienia i jeden z gracz żąda wznowienia zatrzymanej gry, jego przeciwnik nie może odmówić i ma prawo zagrać jako pierwszy.

9. Po uzgodnieniu, że gra się skończyła, każdy gracz usuwa ze swojego terytorium wszystkie kamienie przeciwnika i dodaje je do swoich jeńców. Następnie jeńców ustawia się w terytorium przeciwnika, po czym podlicza się i porównuje punkty terytorium. Gracz z większym terytorium wygrywa.

10. W dowolnym momencie gry gracz może ją zakończyć poprzez przyznanie się do przegranej.

Wymagania funkcjonalne dotyczące iteracji 1.

W iteracji 1 należy zaimplementować następujące wymagania funkcjonalne (**tylko takie !!**):

1. System powinien działać w oparciu o architekturę klient-serwer. W razie wątpliwości warto przyjrzeć się przykładom z tej strony.

2. Gracz za pomocą aplikacji klienckiej powinien móc połączyć się z serwerem i dołączyć do gry.

3. Gracz powinien móc wysłać ruch do drugiego gracza. W tej wersji interfejs użytkownika powinien być konsolowy. Z poziomu klienta (na konsoli) należy wpisać ruch, który będzie wysyłany przez serwer do drugiego klienta. Do tego celu potrzebna będzie klasa przechowująca planszę np. **Board**.

4. Wymagane jest zaimplementowanie zasad 1-3 z powyższej listy.

Wymagania funkcjonalne dotyczące iteracji 2.

W iteracji 2 należy zaimplementować następujące wymagania funkcjonalne (**tylko takie !!**):

1. Wymagane jest zaimplementowanie zasad 4-10 z powyższej listy.

2. Należy stworzyć interfejs okienkowy dla graczy (np. JavaFX)

3. Należy stworzyć diagram klas oraz ewentualnie inny diagram UML obrazujący funkcjonalność aplikacji.

4. Należy wygenerować dokumentację w JavaDoc.

5. Poprawność kodu powinna być na bieżąco weryfikowana testami jednostkowymi napisanymi przy użyciu JUnit

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.

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

Punktacja

Za implementację tej części każdy student może dostać maksymalnie **140 punktów**.

Sposób zaliczenia

Pokazanie rozwiązania prowadzącemu oraz wrzucenie na ePortal: linku do githuba zawierającego wszystkie pliki wraz z instrukcją uruchomienia projektu.