Technologie programowania - 2019
Lista 4

Gra w go

Zaprojektuj i napisz system do przeprowadzania rozgrywek w grę w go. Podstawowe zasady gry i wymagania dotyczące systemu są przedstawione poniżej. Zadanie należy rozwiązać w parach wykorzystując repozytorium. Historia zmian w repozytorium będzie podstawą do oceny wkładu członków zespołu.

Zasady gry

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. 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).

4. 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).

5. 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).

6. 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).

7. 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.

8. 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.

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

Wymagania dotyczące systemu

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, 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.

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 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). Wstępna wersja programu powinna zostać zaprezentowana co najmniej tydzień przed ostatecznym terminem oddania. Ostateczne oddanie projektu wymaga obecności obu członków zespołu.