Piotr Rzeźnik

Wyjaśnienie kilku metod i klas, które pochodzą z symulacji życia miasta górniczego.

Klasa Mine

  
  
1. Zadaniem klasy Mine jest wykonywanie pracy przez krasnoludy, efektem tego jest aktualizacja statystyk wydobycia i stanów szybów kopalni oraz pól krasnoludów.

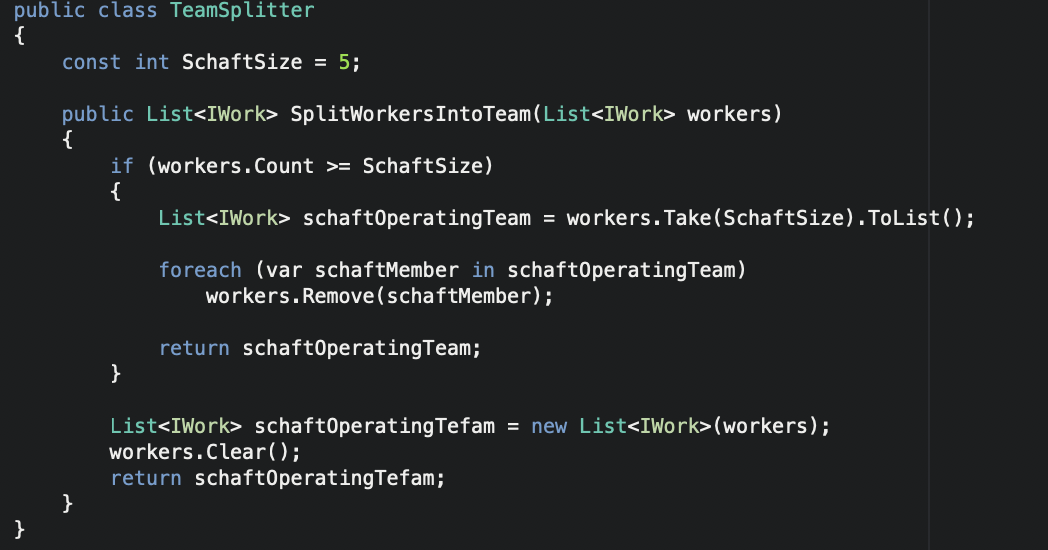
2. Metoda PerformMining przyjmuje parametry w postaci listy obiektów allWorkers, które implementują interfejsy IWork oraz obiekt \_presenter implementujący interfejs IOutputWriter. Metoda jest wywoływana tylko raz, w trakcie jej wywołania jest uruchomiona petla while, która działa do czasu, aż skończą się obiekty z listy, które posiadają pusty plecak oraz ich status życia jest true. Zostaje stworzona lista workers, która posiada tylko te obiekty, których statusy odpowiadają warunkom w konstrukcji LINQ. Następnie w pętli foreach do każdego szybu, który jest sprawny, czyli posiada status Operational. Zostaje mu przypisana lista 5 pracowników za pomocą metody SplitWorkersIntoTeam z klasy TeamSplitter, gdzie będą wykonywać swoje strategie pracy. Po wykonaniu pracy przez wszystkie obiekty w szybach(\_schafts) wracają one do listy workers z zaktualizowanymi stanami. Kolejna pętla foreach sprawdza dla każdego szybu czy został on zniszczony, jeśli tak to jest używany logger(\_presenter) w celu wyświetlenia komunikatu na ekran. Następnie sprawdzane jest czy wszystkie obiekty w szybie zwracają true używając metody GetIsAliveStatus(). W pozytywnym wypadku List<E\_minerals> Stats zostaje uzupełniona o nowa zawartość, którą wydobyły obiekty w szybie. Finalnie odbywa się wyczyszczenie statystyk szybów oraz ich obiektów, które są odpowiedzialne za wykonywanie pracy.

3. Metoda UpdateStats sluzy do zaktualizowania listy allExtractedMaterials, która znajduje się w bycie nadrzędnym, a następnie czysci chwilowe statystyki Stats w klasie Main

4. Metoda FixAllSchafts w każdej następnej turze naprawia wszystkie uszkodzone szyby górnicze (ze statusem Broken)

5.Zaleta rozwiązania, które zastosowałem jest możliwość jego łatwego przetestowania. Elemnty, które implementują interfejsy mogę zamockowac w testach, przez co spodziewam się określonych wartości z randomizerow. Dzięki liście list<IWork> obiekty w której się w niej znajdują, mogą wykonywać jedynie swoja prace na różne sposoby. W ten sposób nie ma możliwości wykonania innych czynności na obiektach poza jego pracą, dzięki temu nie muszę się martwic, że ktoś jak będzie modyfikował tą metodę będzie miał dostęp do pól do których nie powinien mieć dostępu. Jest to elastyczne rozwiązanie, bo niezależnie jaka strategia pracy zostanie zaimplementowana wcześniej w obiekcie, to będzie ona działać w tej metodzie. Dodatkowo, jeśli przyjmuje obiekt typu IRandomizer w konstruktorze klasy Mine, to musze go tylko raz przyjąć właśnie w tym miejscu, polepsza to czytelność i ułatwia testowalność.

Klasa TestSplitter



Zadaniem tej klasy jest tworzenie nowej krasnoludów, która zostanie przydzielona następnie do szybu górniczego. Zastosowałem takie rozwiązanie, ponieważ jest ono elastyczne i pozwala na tworzenie „list robotników” o rożnej ilości. Metoda SplitWorkersIntoTeam przyjmuje listę obiektów, które implementują interfejs IWork. Jeśli liczba krasnoludów jest większa od stałej, która określa pojemność szybów, to wtedy wybiera z listy workers określoną liczbę pracowników i tworzy nową listę, która zostanie przypisana do szybu górniczego. Następnie głównej listy workers są usuwane obiekty, które zostały wybrane do nowej listy w celu przypisania do szybu. Jest to elastyczne rozwiązanie, bo tej klasy nie obchodzi nic poza ilością obiektów w liście, przez co jeśli zmieni się „wnętrze” obiektów, to nie zniszczy to działania tej klasy.

Test do klasy Mine



1.Test PerformMiningAndEveryWorkerGets2MithrilsIntoBackpack służy do sprawdzenia, czy proces wydobycia przebiega zgodnie z założeniami. W given używam zamockowanych obiektów, które domyślnie są randomizerami, pozwala mi to na to, ze przy wywołaniu metod GenerateRandomFromRange i RandomWorkIteration zwrócą mi wartości, których oczekuje. Będą bardziej dokładnym, za każdym razem otrzymam urobek typu Mithril oraz ile sztuk urobku wykopie obiekt. Tworze obiekt mine, gdzie w konstruktorze przesyłam zamockowany obiekt. Tworze listę workers, która posiada 7 obiektów (krasnoludów) typu ojciec, czyli posiadają normalna strategię pracy. Wtedy wywoływana jest metoda PerformMining z listą workers i mockiem. W asercjach sprawdzam czy krasnoludów nadal jest siedem, czy w szybach nie ma żadnych obiektów po wykonaniu pracy, czy w każdym plecaku krasnoluda są dwie jednostki Mithrilu. Właśnie w tym miejscu, dzięki temu ze wcześniej użyłem interfejsów byłem w stanie łatwo przetestować metodę, która opiera się na wartościach losowych i w ogóle ją przetestowac.