SKŁAD GRUPY: **MATEUSZ MOGIELNICKI, TOMASZ MARCHELA, DOMINIK MIERZEJEWSKI**
GRUPA PS: **9**

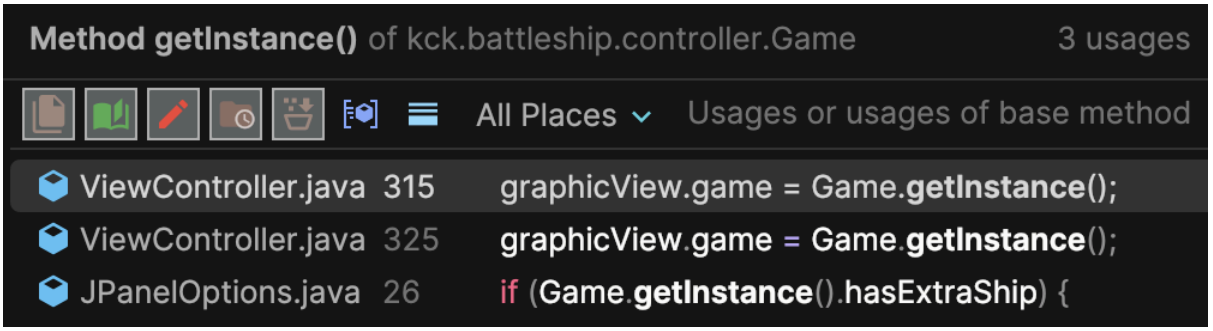
ARCHITEKTURA PROJEKTU – BATTLESHIPS

Wzorce

- **Singleton**

1. Cel użycia

Zapewnienie istnienia tylko jednego obiektu kontrollera klasy Game reprezentującego logikę gry w statki, który jest pobierany tylko przez klasę ViewController wzorca MVC.



Metoda getInstance() jest używana tylko 3 razy z czego 2 razy przez ViewController oraz raz w klasie JPanelOptions do sprawdzenia czy gracz ma dodatkowy statek (co można pominąć).

2. Przyporządkowanie klas do ról wzorca

Klasa Game jest singletonem.

3. Lokalizacja wzorca w kodzie

a. Definicja Singletona w katalogu głownym w controllerach w w pliku Game.java od linii 29 do 37.
b. Stworzenie obiektu w i przypisanie jego referencji do instancji pola game w klasie GraphicView.java poprzez klase ViewController.java.

- **State**

1. Cel użycia

Jest używany do zarządzania różnymi stanami gracza w grze, takimi jak atakowanie(ShotState) czy obrona (DefendState).

2. Przyporządkowanie klas do ról wzorca

Kontekstem stanu jest klasa Player. Stany ShotState, DefendState implementują interfejs PlayerState gdzie dla każdego stanu jest zaimplementowana inna logika w metodach takich jak: defend(), shot(). W głównej mierze stany gracza są zmieniane oraz wykorzystywane w kontrolerze Game w metodzie playTurn().

3. Lokalizacja wzorca w kodzie

   W głównym pakiecie w folderze model.clases.State. | Wszystkie Klasy

4. Wektor Zmian

   Umożliwienie łatwego dodawania nowych stanów bez konieczności modyfikacji istniejącego kodu. Istnieje możliwość dodania kolejnych stanów, takich jak ukrywanie się, czekanie czy ładowanie dział, w zależności od rozwoju logiki gry.

- **Metoda Fabrykujaca**
  1. Cel użycia

     Wzorzec jest używany do tworzenia różnych typów statków w grze.

  2. Przyporządkowanie klas do ról wzorca

     Twórca: ShipFactory
     Konkretny Twórca: (SubmarineFactory, CruiserFactory, DestroyerFactory, BattleshipFactory)

  3. Lokalizacja wzorca w kodzie

     Folder model.clases.FactoryMethod | Wszystkie Klasy

  4. Wektor Zmian

     Wzorzec umożliwia łatwe dodawanie nowych typów statków do gry poprzez dodanie nowych klas "Konkretnego Twórcy" bez konieczności modyfikacji istniejącego kodu.

- **Dekorator**
  1. Cel użycia

     Naliczanie punktów do rankingu gracza w zależności od wybranego poziomu trudności komputera.

  2. Przyporządkowanie klas do ról wzorca

     Komponent: IRanking

     Konkretny Komponent: Ranking

     Dekorator: RankingDecorator

     Konkretny Dekorator: EasyRankingDecorator, NormalRankingDecorator, HardRankingDecorator

3. Lokalizacja wzorca w kodzie

   Folder model.clases.Decorator | Wszystkie Klasy
   Folder model.clases | Klasa Ranking

4. Wektor Zmian

   Wzorzec umożliwia łatwe dodawanie nowych metod naliczania punktów poprzez tworzenie nowych klas bez konieczności modyfikacji istniejącego kodu.

- **Obserwator**
  1. Cel użycia

     Obserwowanie zmian zachodzących na planszy oraz zmian w ilości statków dostępnych na planszy. Obserwatorami jest Probability iterator który w razie zmian przeprowadza iteracje jeszcze raz. Oraz probability table która na podstawie zmian wylicza inne prawdopodobieństwo.

  2. Przyporządkowanie klas do ról wzorca

     Probability Table, Probability iterator

  3. Lokalizacja wzorca w kodzie

     Folder model.clases.strategy | Klasa Observer oraz Hard

     Folder model.clases.iterator | Wszystkie klasy

  4. Wektor Zmian

     Zmiana ilości statków oraz niedostępnych pól.

- **Strategia**
  1. Cel użycia

     Kapsułkacja algorytmów poziomu trudności ai umożliwiająca w prosty i czytelny sposób zaimplementowania poziomów trudności poprzez wprowadzenie odpowiednich algorytmów

  2. Przyporządkowanie klas do ról wzorca

     Diffculty, Easy, Normal, Hard

  3. Lokalizacja wzorca w kodzie

     Folder model.clases.strategy | Wszystkie klasy

  4. Wektor Zmian

Algorytm gry używany przez AI.

- **Iterator**

  1. Cel użycia

     W łatwy sposób zwracanie wszystkich największych prawdopodobieństw z klasy Probability table

  2. Przyporządkowanie klas do ról wzorca

     ProbabilityTable, ProbabilityIterator

  3. Lokalizacja wzorca w kodzie

     Folder model.clases.iterator | Wszystkie klasy

  4. Wektor Zmian

     Element wskazywany przez iterator.

- **Model-View-Controller**

  1. Cel użycia

     Stworzenie graficznego interfejsu użytkownika dla aplikacji i jego oddzielenie od modelu danych wraz z logiką.

  2. Przyporządkowanie klas do ról wzorca

     a. Kontrolery:

        i. Kontroller Game, który zarządza logika gry podczas bitew.

        ii. Kontroller ViewController odpowiada za przełączanie między widokami gdy użytkownik kliknie w kontrolkę. (wywołuje metody z GraphicView oraz logikę z controllera Game)

     b. Modele

        i. Katalog clases - główne klasy gry oraz foldery z klasami wzorców.

ii. Katalog data gdzie są przetrzymywane informacje dotyczące rankingu oraz graczy, którzy coś zakupili w sklepie.

iii. Katalog types - typy pól, statków oraz kierunków.

c. Widoki

i. Interfejs View jest implementowany przez główną klasę GraphicView. Zawiera ona metody, w których zarządza się widocznością ramek czy paneli w widokach.

ii. Oprócz głównej klasy view istnieją również ramki oraz panele wykorzystywane przez GraphicView.

3. Lokalizacja wzorca w kodzie

a. Kontrolery znajdują się w głównym pakiecie w folderze controller, a w nim: kontroller Game oraz ViewController.

b. Klasy, dane, foldery z wzorcami znajdują się w głównym pakiecie w folderze model, a w nim katalogi: clases, data, types.

c. Widoki znajdują się w głównym pakiecie w folderze view, gdzie istnieje interfejs View oraz katalog GrahpicView w którym znajduje się klasa GraphicView oraz wszystkie ramki oraz panele.

d. Wszystkie zdjecia znajduja sie w src.main.resources.

4. Wektor Zmian

Dzięki użyciu wzorca Model-View-Controller można dodać do aplikacji nowe widoki interfejsu użytkownika oraz zmieniać reakcje kontrolek widoku na działania użytkownika poprzez dodawanie nowych obiektów przechowujących nowy kod. Wzorzec ten ułatwia również zmiany modelu i logiki danych, ponieważ są one oddzielone od interfejsu użytkownika.

Specyficzne rozwiązania

Wykorzystanie biblioteki Swing

- Prosta integracja biblioteki Swing do obsługi interfejsu graficznego, co pozwala na interaktywność w grze oraz zapewnia czytelność i estetykę.
- Biblioteka została wykorzystana do wyświetlania wszystkich widoków, oferując funkcje umożliwiające sprawną prezentację elementów graficznych.

Architektura MVC (Model-View-Controller)

- Uproszczenie procesu zmiany lub dodania nowych strategii prezentacji, nie zakłócając podstawowej logiki aplikacji.
- Dzięki zastosowaniu architektury MVC, dodanie nowych widoków w przyszłości jest znacznie ułatwione.

Źródła

- Java SE Application Design With MVC
- Trail: Creating a GUI With Swing (The Java™ Tutorials)
- Design Patterns

Podział Pracy

Mateusz Mogielnicki

- Singleton
- State
- Model-View-Controller

Tomasz Marchela

- Obserwator
- Strategia
- Iterator

Dominik Mierzejewski

- Metoda Fabrykujaca
- Dekorator

Instrukcja Użytkownika

Projekt jest interaktywną wersją klasycznej gry planszowej “Statki”, w której stoczymy pojedynek z przeciwnikiem lub komputerem.

Funkcjonalności

Gra przeciwko Komputerowi

- Gracze mają opcję rozpoczęcia rozgrywki przeciwko komputerowi zaprogramowanym przez programistę.

- Możliwość interaktywnego rozmieszczania statków na planszy przed rozpoczęciem rozgrywki.
- Komputer prowadzi swoją flotę, a gracz musi zastosować swoje umiejętności taktyczne, aby go pokonać.
- Po każdym strzale gra informuje gracza o wyniku.
- Gracz, który jako pierwszy zatopi wszystkie statki przeciwnika, zostaje uznany za zwycięzcę.

Symulacja Gry

- Między Komputerami Możliwość symulacji rozgrywki między dwoma komputerami.

Zasady Gry

- Dostęp do zasad gry w trakcie rozgrywki, umożliwiający łatwe zrozumienie reguł.

Zakupy w Sklepie

- Możliwość zdobycia specjalnych statków lub barier poprzez zakupy w wirtualnym sklepie.

Ranking

- System rankingowy pozwalający śledzić postępy i porównywać umiejętności z innymi graczami.

Instrukcja Instalacji

- Pobierz pliki źródłowe gry.
- Uruchom dany projekt w swoim edytorze kodu.
- Wejdź w plik BattleShips.java.
- Uruchom plik.
- Wybierz w terminalu tryb wyświetlania.
- Postępuj zgodnie z instrukcjami na ekranie.