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

ARCHITEKTURA PROJEKTU – BATTLESHIPS

OPIS PROJEKTU

Projekt gry "Battleships" w Javie to cyfrowa adaptacja klasycznej gry planszowej, w której gracze rozmieszczają flotę statków na siatce i próbują zatopić flotę przeciwnika. W tej wersji Java, gra posiada interfejs użytkownika stworzony z wykorzystaniem Java Swing. Centralnym punktem interfejsu są dwie siatki 10x10, jedna dla gracza, a druga dla komputera, które reprezentują plansze do gry. Gracz może interaktywnie rozmieszczać swoje statki na swojej planszy, a następnie, w trakcie gry, wybierać pola na planszy przeciwnika do "strzału". Wyniki gracza są zapisywane w rankingu. Użytkownik może poprzez punkty w rankingu kupić w sklepie statki oraz bariery na jedna gre. Jeśli gracz nie ma pojęcia jak grać, może wejść w zakładce zasady gry, gdzie wszystko jest krok po kroku objaśnione.

Główne założenia

- ***Cel projektu:*** Przeniesienie klasycznej gry planszowej "Battleships" oraz zastosowanie 8 wzorców takich jak:
    - Singleton
    - State
    - Metoda Fabrykująca
    - Memento
    - Pyłek
    - Strategia
    - Iterator
    - Model - View - Controller

- ***Istota aplikacji:*** Umożliwienie graczom planowania taktyki, rozmieszczania statków oraz skutecznego strzelania w celu zniszczenia floty przeciwnika, przy jednoczesnym wykorzystaniu prostego i intuicyjnego interfejsu graficznego.

Opis Funkcjonalności

1. Gra przeciwko Komputerowi:
   - Gracze mają opcję rozpoczęcia rozgrywki przeciwko komputerowi, zaprogramowanego przez programistę. Przeciwnik będzie posiadał 3 różne poziomy trudności.

   - Gracze będą mieli możliwość interaktywnego rozmieszczania swoich statków na planszy przed rozpoczęciem rozgrywki. Za pomocą prostego interfejsu graficznego będą wybierać pozycję i orientację statków poprzez klikanie strzałek lub poprzez przeciągnięcie statku na plansze.

   - Komputer będzie prowadził swoją flotę, a gracz będzie musiał zastosować swoje umiejętności taktyczne, aby go pokonać.

   - Gra umożliwia na zmianę wykonywanie ruchów, strzelając w konkretne pola na planszy przeciwnika. Po każdym strzale, gra informuje gracza, czy trafienie było udane, czy też nie trafił w wroga.

   - Gracz, który jako pierwszy zatopi wszystkie statki przeciwnika, zostanie uznany za zwycięzcę. Gra wyświetli stosowny komunikat o wyniku, a gracze będą mieli możliwość rozpoczęcia nowej rozgrywki.

   - Na koniec gry zostaną przydzielone punkty w rankingu(tylko w przypadku gdy gracz wygra), gdzie można potem zobaczyć swoje wyniki w odpowiedniej zakładce RANKING w menu.

   - Za punkty w rankingu można kupić w sklepie statek lub barierę.

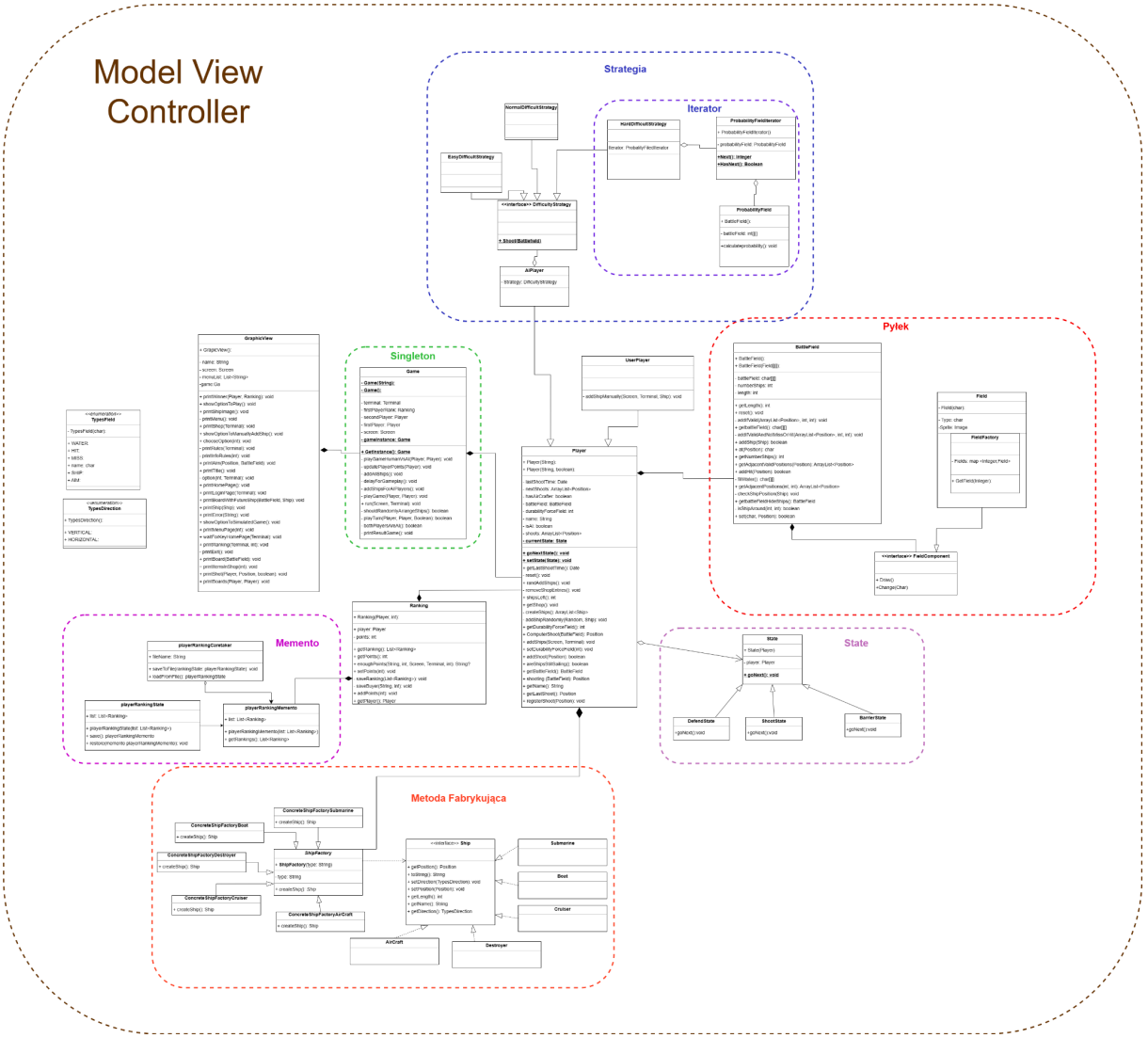
2. Symulacja gry między komputerami:
   - Gra umożliwia również symulację rozgrywki między dwoma komputerami. Algorytm zaimplementowany w programie rywalizuje z drugim takim samym algorytmem, a użytkownik może obserwować przebieg symulowanej bitwy.

3. Zasady Gry
   - Dostęp do zasad gry w trakcie rozgrywki, umożliwiające graczom łatwe zrozumienie reguł oraz specyfiki rozgrywki w dowolnym momencie.

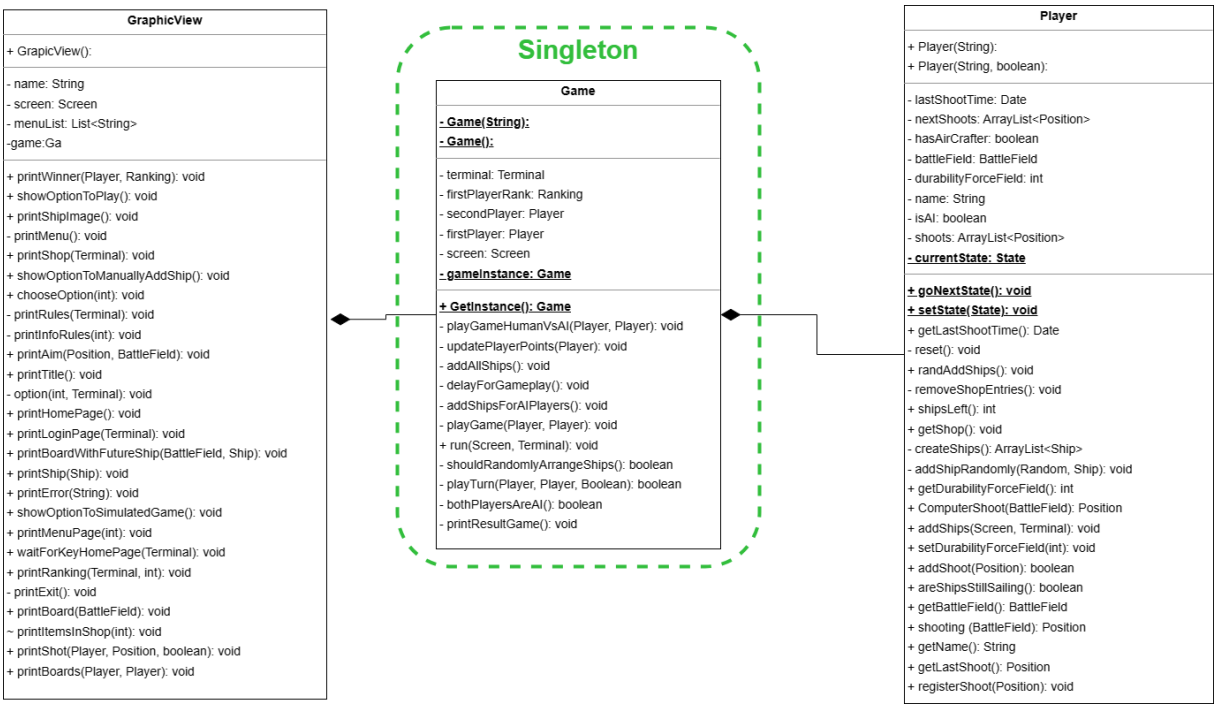
4. Zakupy w Sklepie
   - Gracze będą mieli możliwość zdobycia specjalnych statków lub barier poprzez zakupy w wirtualnym sklepie. To dodatkowe elementy taktyczne, które mogą wpłynąć na przebieg rozgrywki.

5. Ranking
   - System rankingowy pozwala graczom śledzić swoje postępy i porównywać swoje umiejętności z innymi graczami. Wyświetlenie rankingu dostarcza dodatkowej motywacji do poprawy swoich wyników.

Diagram Klas Całego Projektu

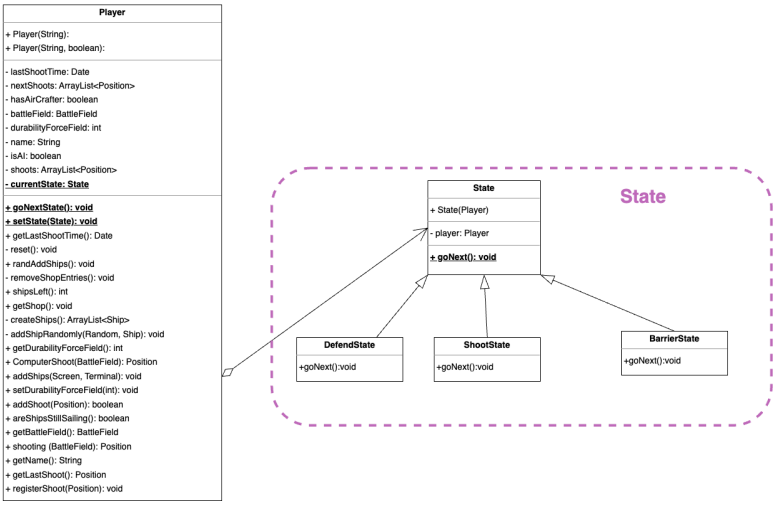


1. Singleton



Singleton wykorzystywany jest w celu zabezpieczenia przed stworzeniem dwóch (lub więcej) instancji Game. Poprzez wykorzystanie prywatnego konstruktora, metody getInstance(). Obiekt jest tworzony w przypadku pierwszego odwołania w metodzie getInstance(). W przypadku kolejnych wywołań zwracana jest zmienna.

2. State

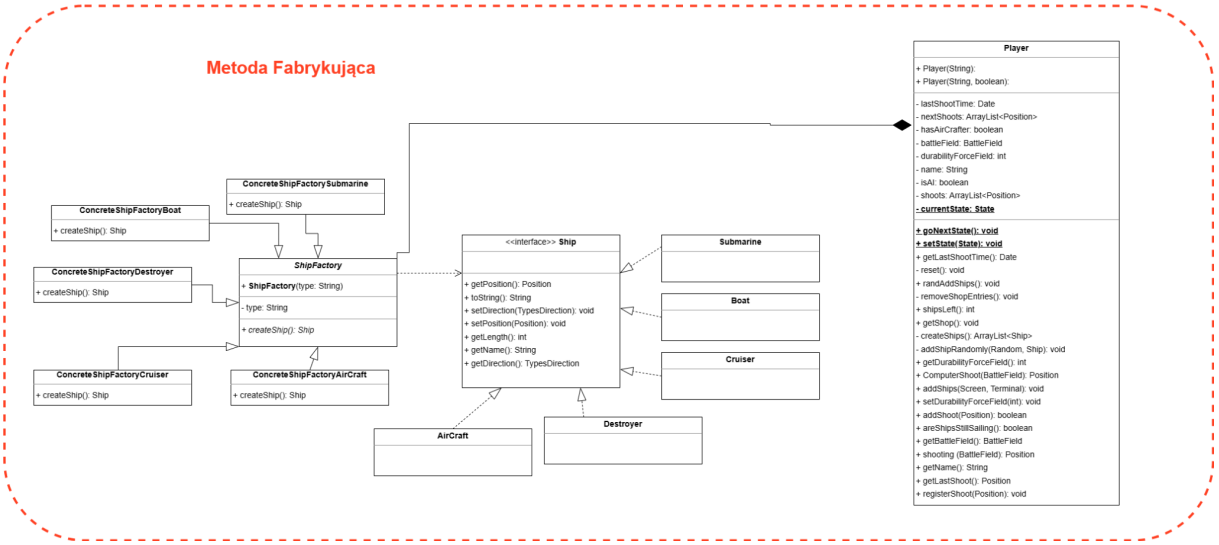


Wzorzec Stan jest używany do modelowania zachowań obiektu, gdy zmienia się jego stan wewnętrzny. Pozwala to obiektom zmieniać swoje zachowanie poprzez przełączenie do innego stanu. Klasa abstrakcyjna State definiuje wspólny interfejs dla wszystkich stanów.

Możemy zobaczyć jedną metodę goNext(), która implementuja inne klasy State. Concrete States (DefendState, ShootState, Barrier State): To konkretnie zaimplementowane klasy stanów, które dziedziczą po State i przedefiniowują jego metody. Każdy z tych stanów reprezentuje różne możliwe stany, w jakich może znajdować się Player: DefendState: Może reprezentować standardowe zachowanie gracza, takie jak obrona przed strzałem. Shoot State: Reprezentuje zachowanie gracza w momencie oddawania strzału. Barrier State: Może symbolizować stan obronny gracza, np. kiedy używa tarczy.

Każdy stan jest odpowiedzialny za zarządzanie logiką związaną z danymi zachowaniami gracza i może decydować o przejściu do innego stanu. Dzięki temu złożone logiki zachowań są rozdzielone i zarządzane przez poszczególne stany, co ułatwia zarządzanie kodem i jego rozszerzanie.

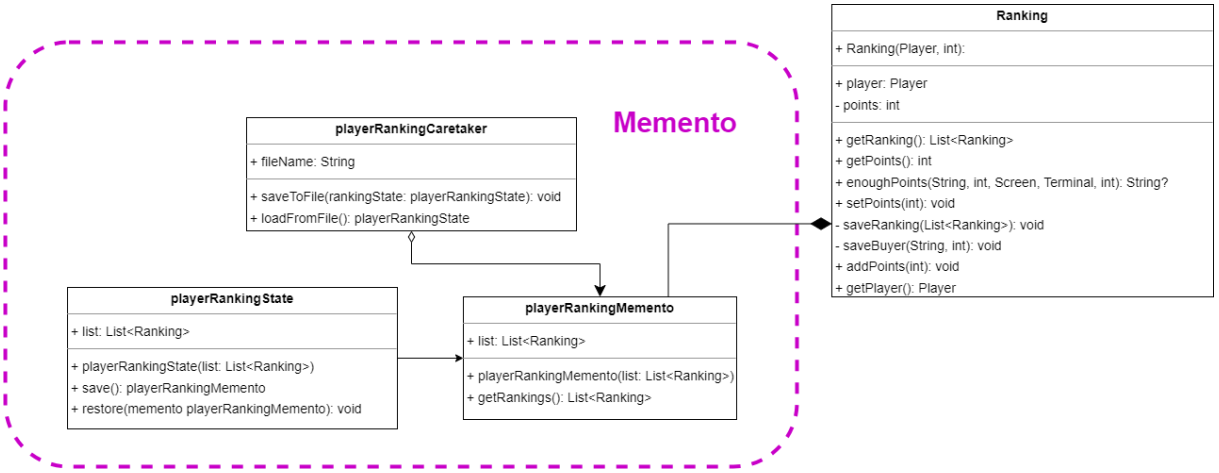
3. Metoda Fabrykująca



Klasy ConcreteShipFactory* implementują metodę createShip(), która jest odpowiedzialna za tworzenie różnych typów statków. ShipFactory jest to abstrakcyjna klasa z metodą createShip(), która musi być zaimplementowana przez wszystkie “Fabryki”, ponieważ to ta metoda decyduje o tym jaki statek ma być stworzony. Ship jest to bazowy interfejs dla różnych typów statków, który zawiera wspólne metody i właściwości.

Klasy dziedziczące z interfejsu to konkretne klasy statków, które implementują specyficzne dla siebie zachowania i właściwości. Player natomiast używa fabryk do tworzenia konkretnych statków. Dzięki wykorzystaniu wzorca Metoda Fabrykująca możemy oddzielić logikę tworzenia obiektów od ich wykorzystania, co pozwala na większą elastyczność i rozszerzalność kodu.

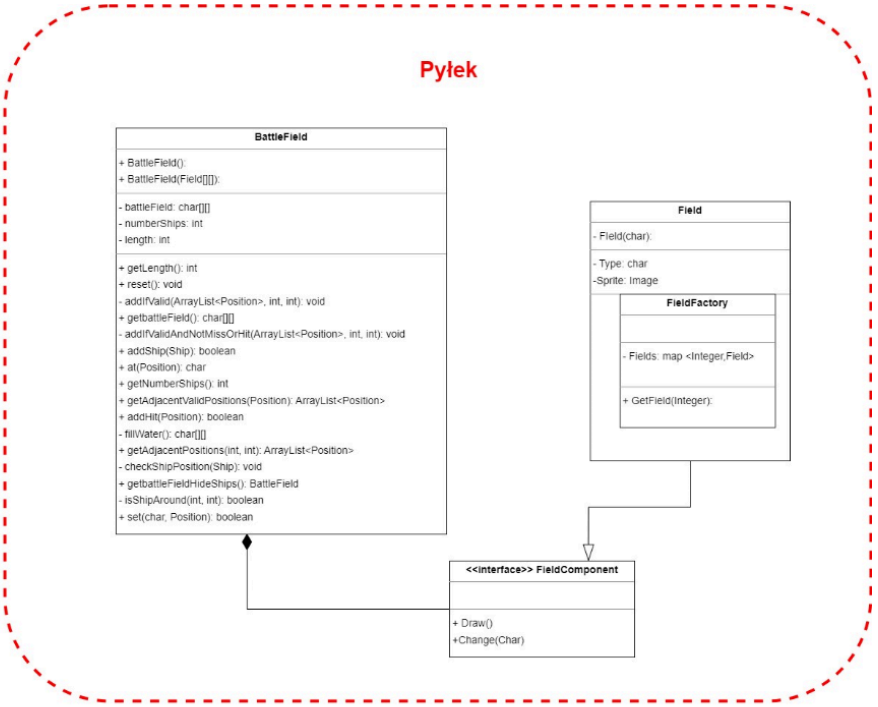
4. Memento



Wzorzec Memento pozwala na zapisywanie stanu obiektu w taki sposób, aby można było później przywrócić ten stan. Klasa playerRankingCaretaker działa jako opiekun (caretaker), który jest odpowiedzialny za zewnętrzne mechanizmy zapisywania i odczytywania stanu, nie wiedząc nic o wewnętrznej strukturze Mementa.

Klasa playerRankingState reprezentuje stan, który ma być zapisany. Zawiera listę rankingów i metody do manipulacji stanem: save() do zapisywania stanu do Mementa i restore() do przywracania stanu z Mementa. Klasa playerRankingMemento jest to Memento, które przechowuje stan playerRankingState. Zawiera kopię listy rankingów, co pozwala na przywrócenie stanu.

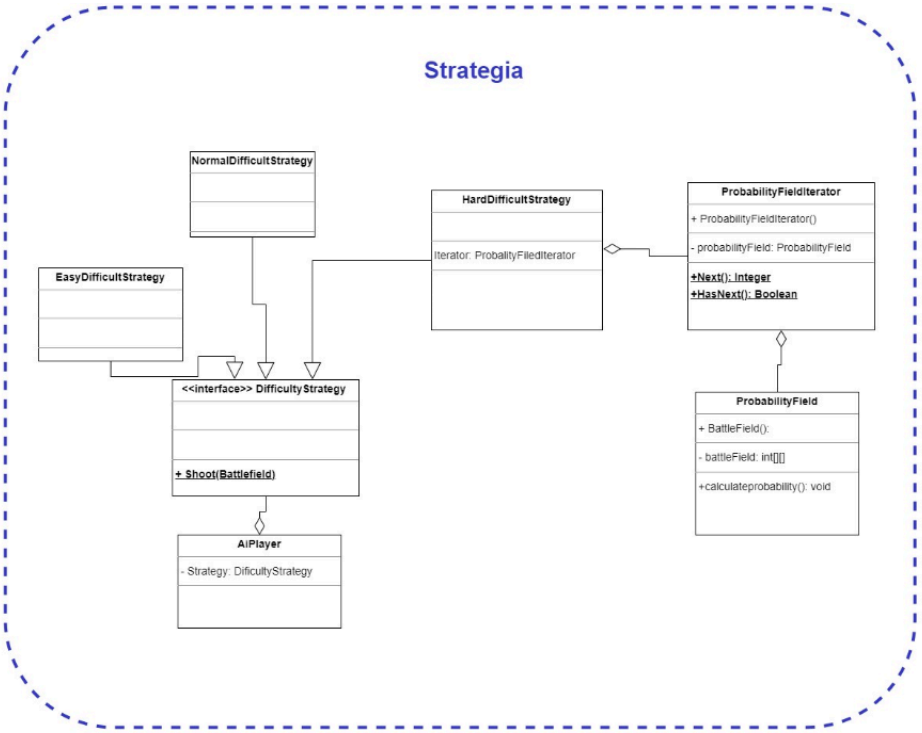
5. Pyłek



Plansza gry jest zbudowana z różnorodnych pól, które często charakteryzują się powtarzalnością, zarówno pod względem typu, jak i reprezentacji graficznej (elementem zewnętrznym). Dostępne typy pól obejmują: wodę, chybiony strzał, fragment statku, trafiony statek oraz zatopiony statek.

Ich kluczowym elementem zewnętrznym jest pozycja na planszy (x,y), którą można przekazać na przykład do funkcji rysującej (draw). Wewnątrz klasy Field znajduje się fabryka, która ma za zadanie zwracać odpowiednie pole. Dzięki temu, zamiast posiadać sto różnych obiektów, będziemy przechowywać w pamięci jedynie tyle, ile jest możliwych stanów pola Field.

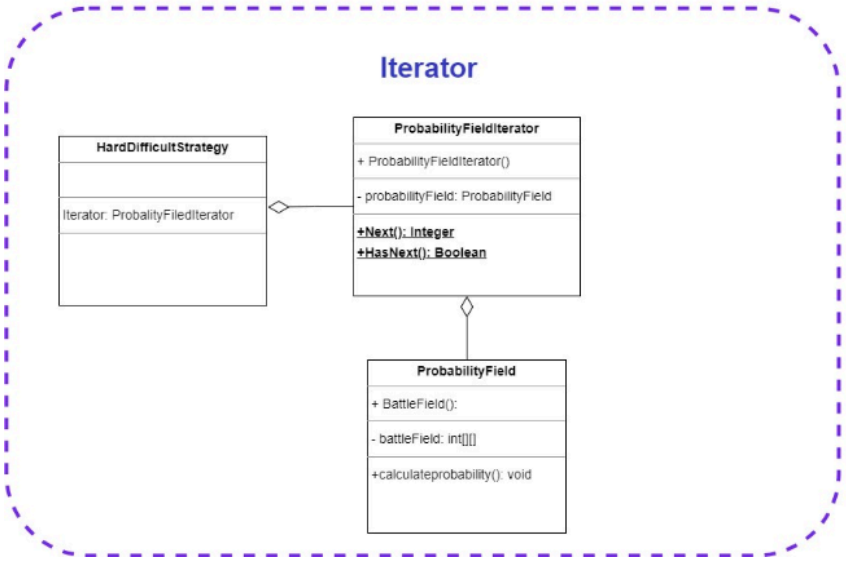
6. Strategia



Przeciwnik z którym się zmierzymy w grze statki będzie miał 3 różne poziomy trudności łatwy, normalny i trudny. Aby zaimplementować owe poziomy zostanie użyty wzorzec “Strategia” do metody odpowiedzialnej za oddanie strzału będziemy przekazywać naszą planszę a algorytm strategii.

Odpowiednio użyte algorytmy do każdego poziomu trudności Easy - przeciwnik strzela losowo jeśli trafi statek szuka kolejnych jego elementów średnia wygrana w 66.2 ruchy, Normal przeciwnik stosuje siatkę czyli jeśli najmniejsze statki jakie pozostały są długości 2 strzela co 2 pola a jeśli pozostały statki o długości 3 strzela co 3 pola średnia wygrana w 55.2 ruchy ostatni poziom trudności hard oblicza tablicę prawdopodobieństwa wystąpienia statku na danym polu i strzela w najbardziej prawdopodobne pole średnia wygrana w 46.2.

7. Iterator



Do ostatniej strategii potrzebujemy uzyskiwać pozycję pola z największym możliwym prawdopodobieństwem. Idealnym narzędziem do tego będzie wzorzec Iterator który będzie przechodził po tablicy i metodą Next() zwracał losowe pole z największym możliwym prawdopodobieństwem i zaznaczał je jako użyte.

Dzięki temu nie będzie potrzebował dostępu do struktury klasy ProbablityField a cała logika strategii będzie łatwiejsza do zrozumienia.

8. Model - View - Controller

Wzorzec MVC to wzorzec projektowy używany do organizacji struktury aplikacji, zwłaszcza w kontekście tworzenia aplikacji z interfejsem użytkownika. Wzorzec ten pomaga w separacji różnych komponentów aplikacji, co ułatwia zarządzanie kodem, utrzymanie i rozwijanie aplikacji. Krótki opis poszczególnych obszarów:

- Model: Odpowiada za reprezentację danych i logiki biznesowej aplikacji. Model jest niezależny od interfejsu użytkownika i widoku. Zawiera logikę związana z dostępem do danych, ich modyfikacją oraz wszelkimi operacjami biznesowymi.

- View: Odpowiada za prezentację danych użytkownikowi. Widok jest odpowiedzialny za wyświetlanie informacji zawartych w modelu użytkownikowi w zrozumiały sposób. Powinien być pasywny i nie zawierać logiki biznesowej ani danych.

- Controller: Zarządza interakcjami użytkownika i przetwarza zdarzenia. Przekazuje żądania użytkownika do modelu, a następnie aktualizuje widok na podstawie odpowiedzi modelu. Kontroler jest łącznikiem między modelem a widokiem.