# UBH - dokumentacja techniczna projektu (przedmiot: Programowanie Obiektowe, Java)

Krzysztof Jajeśnica, nr indeksu 145367, Informatyka, Semestr III

9 grudnia 2020

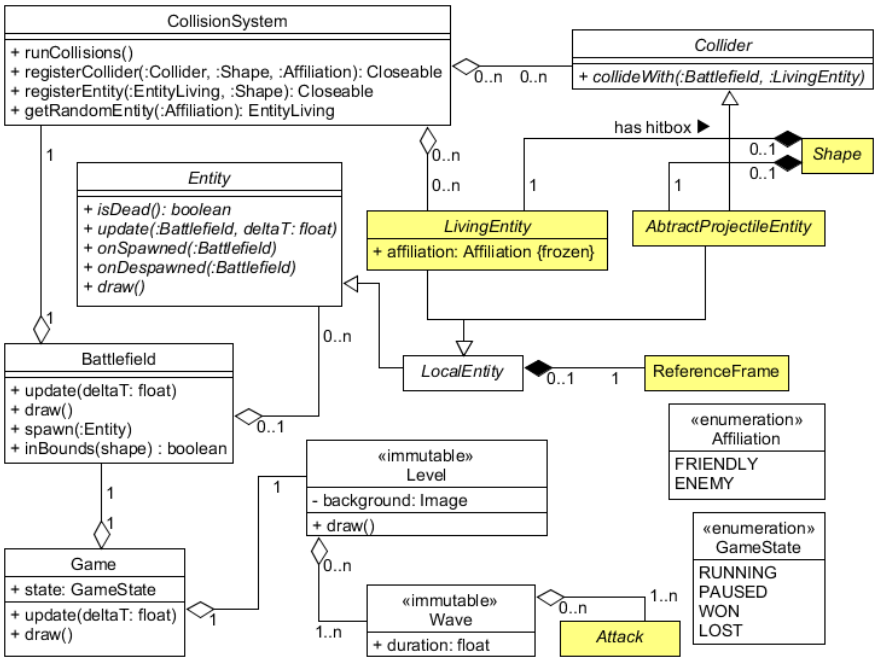
## 1 Zakres projektu

UBH jest grą gatunku *bullet hell* w której gracz lata statkiem kosmicznym i niszczy asteroidy oraz statki kosmitów.

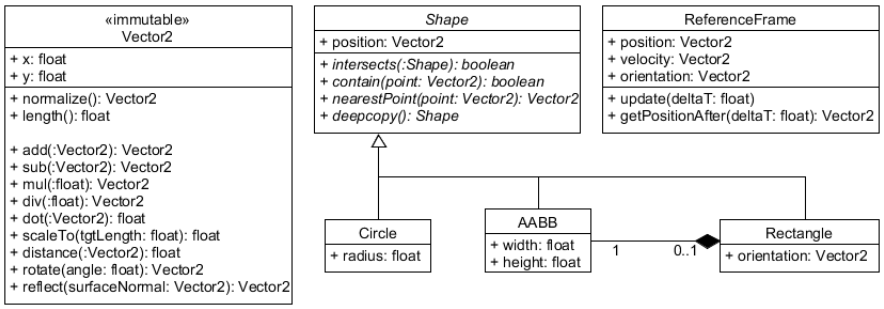
## 2 Diagram klas

Z powodu dużej ilości klas diagram został podzielony na części. W każdej części kolorem żółtym oznaczono klasy pochodzące z innej części diagramu.

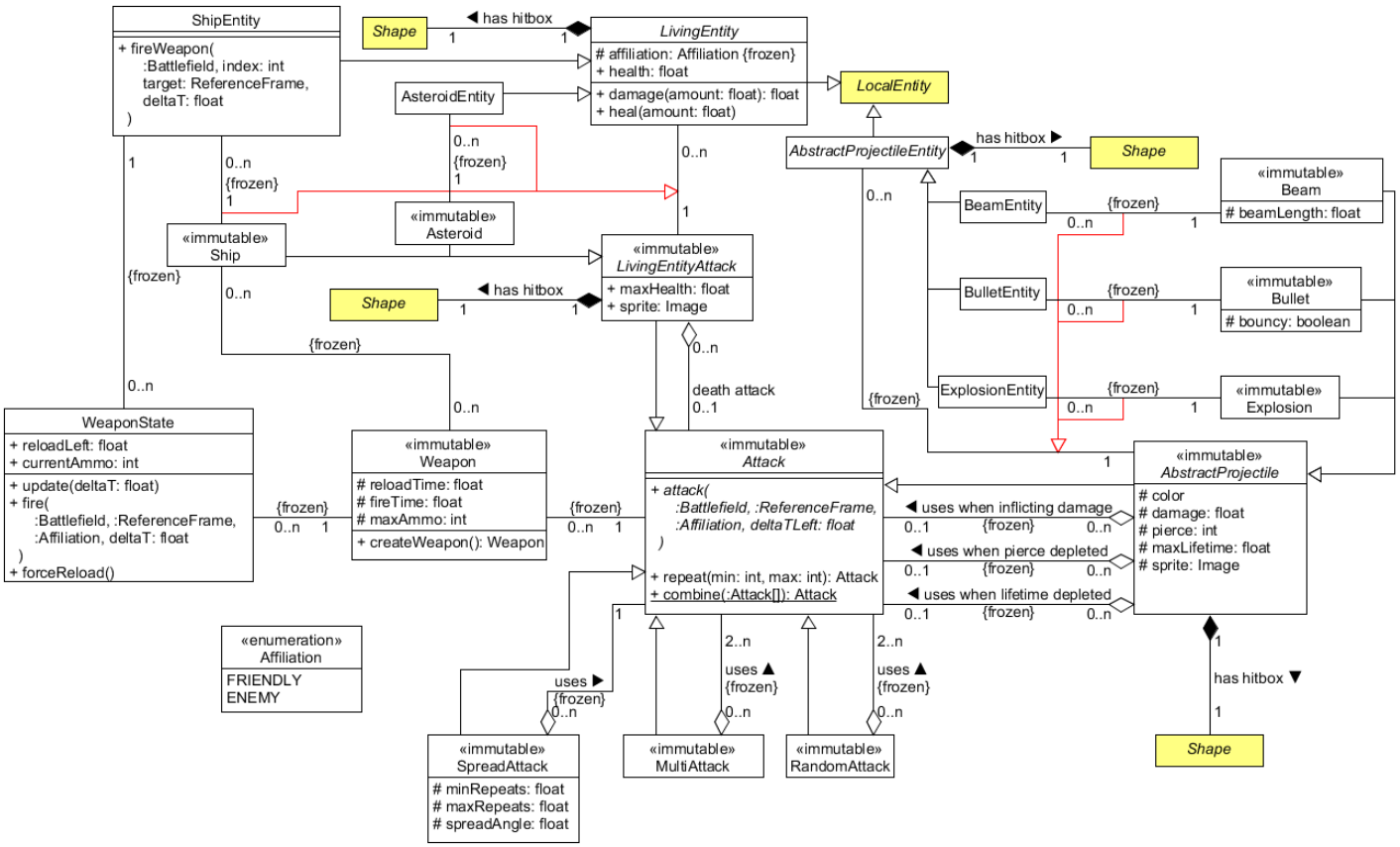
### 2.1 Ogólny plan systemu



## 2.2 Klasy pomocnicze




## 2.3 Ataki




Atak (`Attack`) jest abstrakcją opisującą akcję, którą można wykonać by zadać obrażenia bytom żywym (`LivingEntity`) znajdującym się na polu bitwy.

Atak zawsze jest wykonywany w pewnym układzie odniesienia (`ReferenceFrame`). Argument `deltaTLeft` jest czasem od użycia ataku do końca obecnej klatki. Wszystkie ataki są immutable, aby umożliwić współdzielenie tego samego ataku przez wiele obiektów.

#### 2.3.1 Ataki proste

Ataki proste bezpośrednio tworzą nowe byty na polu bitwy:

- Podklasy `AbstractProjectile` tworzą pociski.
- Podklasy `LivingEntityAttack` tworzą byty żywe (statki, asteroidy, etc.).

#### 2.3.2 Ataki złożone

Ataki złożone składają się z innych ataków i modyfikują sposób ich wykonywania:

- `MultiAttack` łączy wiele ataków w jeden atak.
- `RandomAttack` wykonuje losowy atak z listy.
- `SpreadAttack` wykonuje dany atak wielokrotnie, przy każdym powtórzeniu modyfikując układ odniesienia, aby rozproszyć efekty ataków na większy obszar.

# 3 Wymagania systemowe

## 3.1 Wymagania funkcjonalne

- Gracz może sterować statkiem przy użyciu klawiatury oraz myszki.
  - Statek gracza może poruszać się w górę, w dół, w lewo oraz w prawo, ale nie może opuszczać widocznego pola bitwy.
  - Gracz ma możliwość tymczasowego spowolnienia kontrolowanego statku przez wciśnięcie przycisku.
  - Gracz może kontrolować bronie znajdującej się na statku. Możliwe jest używanie jednej broni na raz. Gracz może wybierać, którą z broni znajdujących się na statku chce używać.
- Gra powinna zawierać wiele poziomów.
  - Poziom powinien posiadać tło w postaci obrazu.
  - Poziom powinien zawierać wiele fal przeciwników.
  - Poziom jest wygrywany w momencie pokonania wszystkich fal przeciwników, a przegrywany w momencie zniszczenia statku gracza.
  - Gracz ma możliwość powtórnego przejścia danego poziomu dowolną liczbę razy.

- Gra powinna zawierać różne rodzaje statków gracza.
  - Każdy statek jest wyposażony w jedną lub więcej broni strzelających pociskami. Pociski zadają obrażenia kontaktowe.
  - Każdy statek ma określony rozmiar, wygląd oraz szybkość lotu.
  - Każdy statek posiada pewną liczbę punktów życia. Wyczerpanie punktów życia skutkuje zniszczeniem statku.
- Gra powinna zawierać różne rodzaje przeciwników.
  - Przeciwnicy mogą być statkami kosmitów lub asteroidami.
  - Statki kosmitów automatycznie atakują statek gracza przy użyciu broni.
  - Asteroidy zadają obrażenia przy kontakcie z graczem.
  - Przeciwnicy mogą używać dodatkowych ataków w chwili zniszczenia (np. eksplodować, stworzyć dodatkowych przeciwników).
- Gra powinna posiadać graficzny interfejs użytkownika.
  - Interfejs użytkownika powinien zawierać menu główne, menu ustawień oraz menu wyboru poziomów.
  - Gra powinna umożliwiać otwarcie menu ustawień podczas gry.
- Gra powinna umożliwiać tymczasowe wstrzymywanie oraz wznawianie gry.
  - Podczas tymczasowego wstrzymania gry gracz ma możliwość przejścia do menu ustawień, zakończenia poziomu lub restartowania poziomu.
  - Wstrzymywanie gry powinno być kontrolowane przy użyciu klawiatury.
- Gra powinna umożliwiać definiownie nowej zawartości.
  - Gracz może definiować nowe rodzaje statków, asteroid, broni oraz pocisków.
  - Gracz może definiować nowe poziomy.
  - Zawartość zdefiniowana przez gracza nie zawiera nowego kodu wykonywalnego.

## 3.2 Wymagania pozafunkcjonalne

### 3.2.1 Tworzenie nowej zawartości przez gracza

Nowe poziomy/bronie/przeciwnicy mogą być definiowane przy użyciu plików w formacie HJSON. Podczas uruchamiania gra parsuje pliki HJSON znajdujące się w ustalonym folderze i tworzy na ich podstawie odpowiednie obiekty. W przypadku napotkania błędnej definicji zawartości gra przerywa ładowanie tej

zawartości, informuje użytkownika o wystąpieniu błędu oraz zapisuje dokładne informacje o błędzie do pliku tekstowego. Zawartość zależna od błędnej zawartości nadal zostaje załadowana, ale z pominięciem zawartości błędnej (przykładowo błąd w definicji statku kosmity zapobiega pojawianiu się tego statku we wszystkich poziomach, w których definicji był on użyty). Użytkownik powinien zostać poinformowany o pominięciu błędnej zawartości.

Przykładowy format pliku:

```
{
    type: Asteroid
    id: small_rock_asteroid
    maxHealth: 100
    contactDamage: 50
}
```

Każdy zdefiniowany obiekt ma pewne ID. Definicje obiektów mogą odnosić się do innych obiektów przez ich ID. Zależności cykliczne są zabronione.

```
{
    type: Asteroid
    id: medium_rock_asteroid
    maxHealth: 500
    contactDamage: 100
    deathAttack: {
        type: SpreadAttack
        minRepeats: 3
        maxRepeats: 5
        spreadAngle: 360deg
        attack: small_rock_asteroid
    }
}
```

### 3.2.2 Praca przy zmiennej liczbie klatek na sekundę

Gra powinna dostosowywać liczbę klatek na sekundę do możliwości sprzętu użytkownika (zakładamy że z pojedynczą klatką związana jest pojedyncza aktualizacja stanu gry - w diagramie klas metoda `update`). Zmienna liczba klatek na sekundę nie powinna powodować nieprawidłowego działania gry, w szczególności:

- Zmiana liczby klatek na sekundę nie powinna powodować zmiany szybkości działania gry (e.g. jeśli dany pocisk eksploduje po 5 sekundach od pojawienia się na polu bitwy, to powinien to robić z tym samym opóźnieniem niezależnie od tego czy jest 20FPS czy 300FPS).
- Częstotliwość strzelania broni powinna być całkowicie niezależna od liczby klatek na sekundę. W szczególności przy strzelaniu z częstotliwością większą od obecnego FPS liczba wystrzelonych pocisków w ciągu sekundy oraz odstępy miedzy pociskami powinny być takie same jak w przypadku FPS większego od częstotliwości strzelania.

Gra powinna działać stabilnie dopóki liczba klatek na sekundę pozostaje powyżej 20FPS.

### 3.2.3 Wieloplatformowość

Gra powinna poprawnie funkcjonować w systemach operacyjnych Windows oraz Linux.

#### 3.2.4 Łatwość instalacji

Wszystkie dodatkowe pliki oraz biblioteki konieczne do poprawnego funkcjonowania gry powinny razem z grą znajdować się w pojedynczym pliku .JAR.

# 4 Realizacja projektu

## 4.1 Plan realizacji systemu

| | |
|---|---|
| Utworzenie działającego prototypu | 2020-12-14 - 2020-12-24 |
| Implementacja wymagań systemowych | 2020-12-25 - 2021-01-07 |
| Końcowe testowanie i dopracowywanie projektu | 2021-01-08 - 2021-01-21 |

## 4.2 Wykorzystane biblioteki i narzędzia

| Biblioteka/narzędzie | Zastosowanie w projekcie |
|---|---|
| hjson-java | Parsowanie plików HJSON. |
| JUnit 5 | Wykonywanie testów jednostkowych. |
| Gradle | Budowanie projektu. |

# 5 Kryteria akceptacyjne

Razem z wersją 1.0.0 gry na platformie GitHub zostanie umieszczony folder skompresowany `test.zip` zawierający materiały służące do weryfikacji wymagań systemowych:

- Skompilowany program `ubh.jar`
- Pliki definiujące dodatkową zawartość: w szczególności `bad_level.hjson` zawierający niepoprawną definicję poziomu, `bad_alien.hjson` zawierający niepoprawną definicję kosmity oraz `good_level_bad_alien.hjson` zawierający poprawną definicję poziomu odnoszącą się do niepoprawnej definicji kosmity.

Procedura weryfikacji wymagań:

1. Uruchomić `ubh.jar`
   - 1.1 Powinien pojawić się komunikat o niepoprawnej zawartości użytkownika.
   - 1.2 W folderze powinien pojawić się plik informujący o błędzie w zawartości `bad_level.hjson` oraz `bad_alien.hjson`.
2. Przejść do menu wyboru poziomów.
   - 2.1 Zweryfikować brak poziomu `bad_level`/informację o błędzie w tym poziomie w liście poziomów użytkownika.

2.2 Zweryfikować obecność poziomów `good_level` oraz `good_level_bad_alien` na liście.

3. Uruchomić poziom `good_level`.

   3.1 Powinno wyświetlić się tło poziomu.

   3.2 Po rozpoczęciu poziomu powinni zacząć pojawiać się przeciwnicy.

   3.3 Należy zweryfikować możliwość poruszania statkiem, ograniczenie możliwości poruszania do krawędzi pola bitwy, możliwość spowalniania ruchu statku, celowania, strzelania oraz zmiany broni.

   3.4 Wstrzymać grę.

   3.5 Zweryfikować możliwość otwarcia menu ustawień oraz powrotu z menu ustawień do rozgrywki.

   3.6 Wznowić grę.

   3.7 Zweryfikować otrzymywanie obrażenia od asteroid oraz pocisków przeciwników przez statek gracza.

   3.8 Pozwolić na zniszczenie statku gracza.

   3.9 Powinna wyświetlić się informacja o przegranej oraz przycisk służący do restartowania poziomu.

   3.10 Użyć przycisku do restartowania poziomu. Zweryfikować powrót pola walki do stanu początkowego (ta sama pozycja statku, napełniony pasek punktów życia, brak przeciwników na ekranie).

   3.11 Zweryfikować możliwość niszczenia kosmitów i asteroid przez pociski gracza. Część kosmitów oraz asteroid powinna wystrzeliwywać dodatkowe pociski po zniszczeniu.

   3.12 Po pokonaniu wszystkich fal kosmitów powininna się wyświetlić informacja o wygranej z możliwością powrotu do menu wyboru poziomów/menu głównego.

4. Uruchomić poziom `good_level_bad_alien`.

   4.1 Powinien wyświetlić się komunikat informujący o brakującej zawartości.

   4.2 Poziom nie powinien zawierać żadnych przeciwników (ponieważ był tak spreparowany że jedyny pojawiający się rodzaj przeciwnika to nieprawidłowy `bad_alien`).

   4.3 Po kilku sekundach powinna pojawić się informacja o wygranej.

Ostateczna wersja procedury weryfikacji wymagań systemowych zostanie opublikowana razem z wersją 1.0.0 programu.