SZKOŁA DOBREGO KODU

10 ZADAŃ NA START DO WŁASNEGO HEROES 3

WSTĘP

Skoro czytasz ten dokument, to zakładam 4 rzeczy:

1. Widziałeś większość z **playlisty Drogowskaz** itd. na YouTube .
2. Zainteresował Cię paradygmat programowania obiektowego.
3. Ufasz mi, że kluczowe jest, aby kod był przetestowany, łatwy w zrozumieniu oraz otwarty na zmiany.
4. Po prostu chcesz ćwiczyć na aplikacje Heroes.

Jak rozpocząć

```
git clone
https://github.com/software-
wizard/HEROES_SAMPLE_APP
```

Wszystko znajdziesz w pliku readme.md

Pełny proces powstawania tego sample wraz ze wskazaniem najważniejszych elementów możesz zobaczyć we wspomnianej **playliście.**

POWODZENIA!

Przydatne linki

YouTube gdzie pokazuję, jak było to tworzone wraz z dokładnym opisem

Opis kreatur, czarów, artefaktów i całej reszty z oryginału

Grafiki z oryginału do wykorzystania

Darmowe(legalne) demo oryginału heroes

Repozytorium z kodem Heroes

Zadanie 1

Refaktoring, hermetyzacja.

Klasa MainBattleController robi zdecydowanie za dużo. By uniknąć późniejszych problemów w implementacji, wynieś ***Map<Point, GuiTileIf> board ,***jako osobną klasę, wraz z metodami move itd.

To samo zrób z klasą:
Queue<Creature> creaturesQueue

Nie zapomnij o testach jednostkowych ;). Dzięki takiemu rozbiciu możemy testować mniejsze elementy.

Zadanie 2

Jednostki latające / chodzące

Aktualna implementacja metody ***move***, w żaden sposób nie sprawdza, czy przed nami są przeszkody typu ***RockObstacle***. Można powiedzieć, że jednostki będą przelatywać / teleportować się przez przeszkodę.

Zatem można powiedzieć, że aktualnie zaimplementowaliśmy jedynie jednostki chodzące.

Zacznij od refaktoringu i użycia wzorca **strategia**, który pomoże Ci rozdzielić 2 typy chodzenia. Stwórz algorytm wykrywania kolizji.

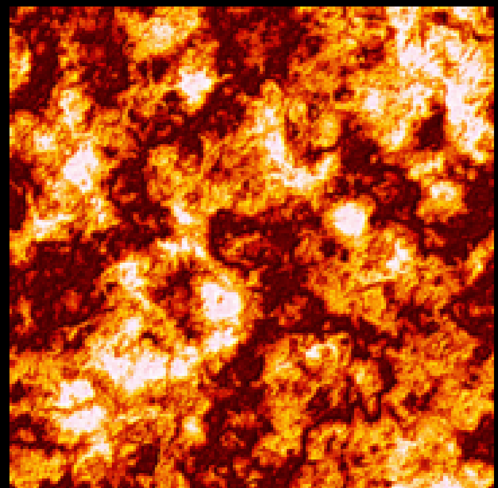
Ważne byś zapamiętywał ścieżkę, którą będą podążać Twoje kreatury. Będzie to potrzebne do kolejnego zadania.

Zadanie 3

Lawa na planszy

Wyświetliliśmy lawę, ale nic ona nie robi.
Skoro masz ścieżkę, którą podąża Twoja kreatura, to teraz na każdym kafelki wywołujesz:
GuiTileIf.apply(creature);
Implementacja w ***Creature***, oraz ***RockObstacle*** powinna być pusta. Natomiast w LavaObstacle powinna zadawać jakieś obrażenia ***Creature***, która przyszła w parametrze ;).

Fajne to programowanie obiektowe gdy mamy poprawny model nie? Tak niewiele kodu, tak wiele możliwości.



Zadanie 4

Błoto na planszy

Teraz gdy masz już ścieżkę, którą podąża kreatura. W jakiś sposób liczysz punkty ruchu. Nie będzie problemem dodanie nowego rodzaju pola, które spowoduje odjęcie kilku punktów ruchu, prawda?

Zadanie 5

Czar magic arrow, zadający np. 100 pkt obrażeń.

No to zaczyna się robić trudno... Dodajemy całkiem nowe zachowanie.
Pamiętaj, by zacząć od testów!

Polecam taki:

```
@Test
void myFirstSpellTestObjectProgrammingIsAwesome(){
    //given
    Creature c = CreatureFactory.create(CreatureFactory.BEHEMOTH);
    Spell magicArrow = new MagicArrow(100);

    //when
    magicArrow.cast(c);

    //then
    assertEquals( expected: 60,c.getCurrentHp());
}
```

Jeszcze tego nie wyświetlaj.
Wrócimy do tego!

Zadanie 6

Czar typu debuff - spowolnienie

Skoro mamy już 1 czar to czas na kolejny!
Po rzuceniu na jednostkę zabiera X punktów ruchu. Jak zawsze zaczynamy od testu.

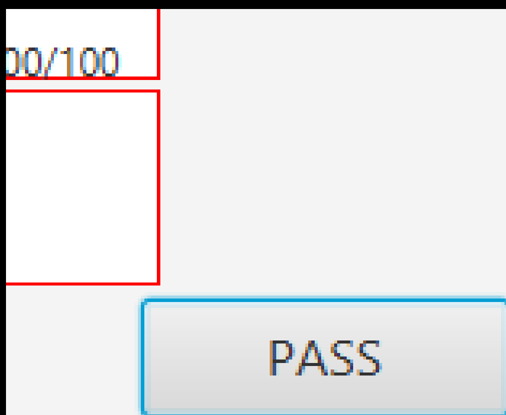
```
@Test
void myFirstSpellTestObjectProgrammingIsAwesome(){
    //given
    Creature c = CreatureFactory.create(CreatureFactory.BEHEMOTH);
    Spell slowDebuff = new Slow();

    //when
    slowDebuff.cast(c);

    //then
    assertEquals( expected: 3,c.getMoveRange());
}
```

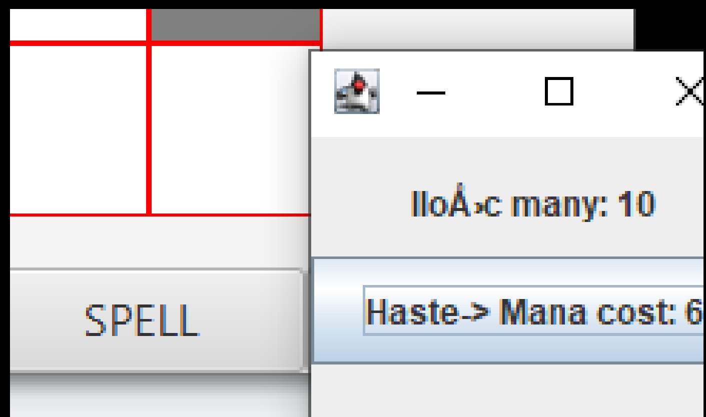
Zwróć uwagę, API obu czarów jest identyczne!
Mają wspólną klasę bazową ***Spell***, a implementacja ***Slow*** oraz ***MagicArrow*** będzie zupełnie inna.

Zadanie 7

Możliwość rzucenia czarów w GUI

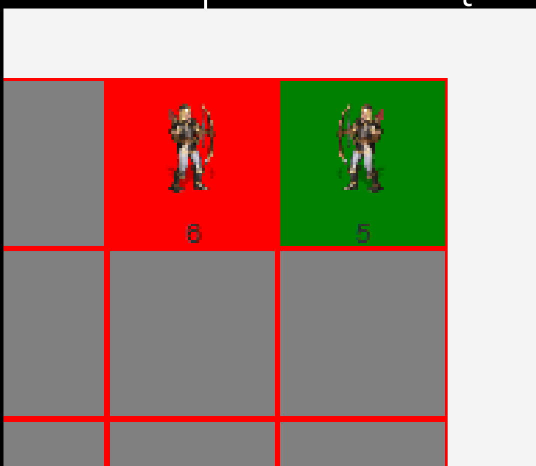
Dodaj tutaj nowy guzik Spells. Na click otwórz okno możliwością wyboru czaru.

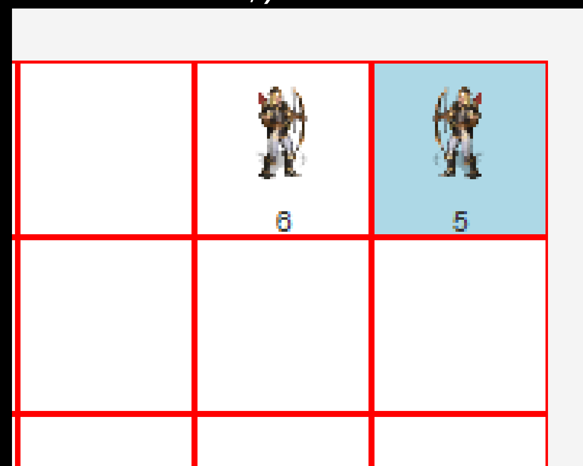




Po wybraniu odśwież gui i zamiast pokazywać zasięg ruchu, podświetl na niebiesko jednostki, na które można rzucić czar. Po kliknięciu jednostki, po prostu wywołaj ***spell.cast(clickedCreature).***

Czas zapoznać się z wzorcem State ;).





Zadanie 8

Połączmy czary i pola

Stwórz nowy typ pola, który po przejściu przez nie będzie nakładał efekt czaru ***Slow***.

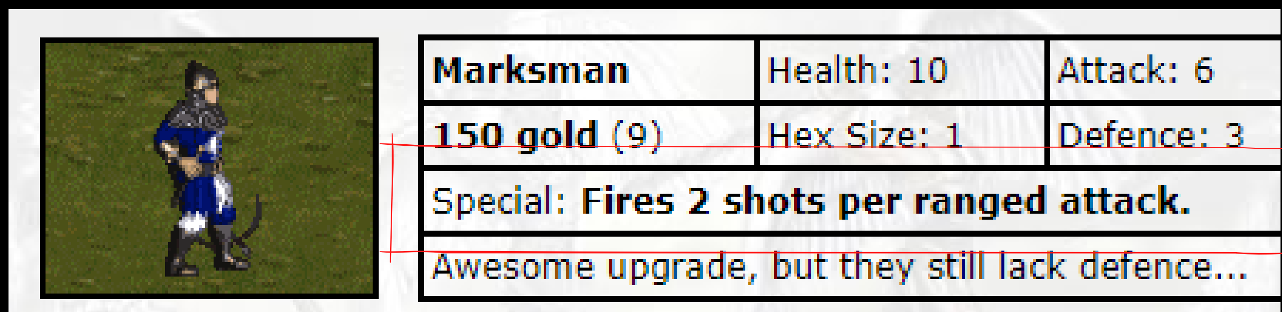
Skoro jesteś już tak daleko, to pewnie masz własny pomysł na implementacje ;). Gratulacje!

Jeszcze tylko 2 zadania.

Zadanie 9

Frakcje

Stwórz 2 frakcje. Odwzoruj je w pełni.
Zwróć szczególną uwagę na własności specjalne np:



Dziedziczenie nie będzie najlepszym pomysłem. Zapoznaj się z wzorcem dekorator. Dekorator i strategia powinny Ci wystarczyć do tego, by stworzyć to w sposób bardzo ładny.

Na koniec zrób jeszcze fabrykę abstrakcyjną, której API będzie wyglądać tak:

```
abstract public Creature create(int aTier, boolean aUpgraded);
```

Konkretna implementacja InfernoFactory czy FortressFactory, będą wiedziały, jakiego ***Creature*** stworzyć.

Zadanie 10

Wprowadź bohatera

Do tej pory wszystkie elementy, jakie stworzyłeś, były oderwane od siebie. Stwórz Bohatera, który będzie wszystko agregował.

Będzie miał listę kreatur, listę czarów, jakieś punkty magii itd.

Jak to zrobisz, możesz dorabiać, co tylko zechcesz. Dorób artefakty, umiejętności specjalne, niech fantazja Cię poniesie!

Oglądaj to co zrobiłeś r**az na 3 miesiące** i zastanawiaj się, co można poprawić, co można zrobić bardziej elegancko.
Z każdą nowo przeczytaną książką. Z każdym nowym odbytym kursem, zerknięcie na kod z przeszłości rodzi wesołe odczucia ;)

Jeśli udało Ci się wykonać powyższe zadania, możesz sobie pogratulować.

```
if(tasks.stream().filter(Task::isDone).count()==10){
    System.out.println(" !!! CONGRATS !!! ");
}
else{
    System.out.println(" GO TO WORK!!! ");
```

```
}
```

Niech wzorce projektowe, SOLID oraz testy jednostkowe na zawsze będą z Tobą! :)