Załącznik 1. Sposób przydziału specjalnej umiejętności Człowieka poszczególnym studentom.

Przydział specjalnej umiejętności do implementacji jest zdeterminowany numerem indeksu oraz inicjałami autora.

Przydział jest realizowany w następujący sposób:

$$ID = X \bmod 5$$

gdzie:

ID – id (wg poniższej tabeli) specjalnej umiejętności Człowieka

X – ostatnia cyfra numeru indeksu

Tabela 4. Specjalne umiejętności Człowieka.

Id	Umiejętność	Cechy
0	Nieśmiertelność	Człowiek nie może zostać zabity. W przypadku konfrontacji z silniejszym przeciwnikiem przesuwa się na losowo wybrane pole sąsiadujące.
1	Magiczny Eliksir	Siła Człowieka rośnie do 10 w pierwszej turze działania umiejętności. W każdej kolejnej turze maleje o „1”, aż wróci do stanu początkowego.
2	Szybkość antylopy	Człowiek w porusza się na odległość dwóch pól zamiast jednego przez pierwsze 3 tury działania umiejętności. W pozostałych 2 turach szansa że umiejętność zadziała ma wynosić 50%.
3	Tarcza Alzura	Człowiek odstrasza wszystkie zwierzęta. Zwierzę które stanie na polu Człowieka zostaje przesunięte na losowe pole sąsiednie.
4	Całopalenie	Człowiek niszczy wszystkie rośliny i zwierzęta znajdujące się na polach sąsiadujących z jego pozycją.

Załącznik 2. Przykładowe błędy za które będzie obniżana ocena

- Brak polimorfizmu i przynajmniej jednej klasy abstrakcyjnej (-1 pkt)
- Nadużywanie funkcji statycznych/luźnych funkcji (-1 pkt)

  W projekcie C++ jedyną luźną funkcją może być main. Proszę nie nadużywać słowa kluczowego ***static*** i używać go tylko tam, gdzie jest to konieczne.
- Brak hermetyzacji (-1 pkt.)

  Pola klas powinny być chronione modyfikatorami ***private*** lub ***protected***. Pola o zakresie widoczności ***public*** mogą być stosowane tylko wtedy, gdy reprezentują wartości stałe (***const***/***final***), najlepiej, aby były statyczne.
- Cechy specjalne organizmów zaimplementowane poza klasą, dla której są właściwe (-1 pkt)

  Przykładowo błędem będzie zaimplementowanie w metodzie **kolizja()** klasy Zwierze sprawdzania czy organizm z którym zderzył się organizm atakujący jest klasy Zolw, a następnie porównywanie siły i pozostawanie na tym samym polu jeśli jest to konieczne. Lepiej dodać metodę wirtualną **bool czyOdbilAtak(Organizm& atakujacy)** do organizmu lub zwierzęcia i następnie wywołać ją w trakcie kolizji. Dzięki temu można także zaimplementować tarczę Alzura.

Załącznik 3. Sugerowane dobre praktyki programistyczne

Klasy i obiekty

- Dobrze jest stosować logiczny podział na przestrzenie nazw - każda przestrzeń nazw w oddzielnym module (pliku).
- Metody które nie wykorzystują obiektu powinny być statyczne. Nie należy ich nadużywać.

Hermetyzacja

Dobrze jest aby wybrane klasy miały metody typu get i set dla składowych lub tylko get, lub całkowity brak dostępu bezpośredniego.

Dziedziczenie

Wielokrotne wykorzystanie kodu (kod w klasie bazowej używany przez obiekty klas pochodnych). Podczas nadpisywania metody klasy bazowej można dalej wykorzystywać implementacje z klasy nadrzędnej (bazowej).

Polimorfizm

- Warto wykorzystać metody służące do dynamicznego określania typu:
  - dynamic_cast<klasa_pochodna*>() dla C++
  - instanceof dla Javy
  - isinsance() lub type(a) is … dla Pythona
- Warto stosować interfejsy oraz klasy abstrakcyjne.

Inne

Warto zastosować wyjątki do sygnalizacji i obsługi błędów.

- http://en.cppreference.com/w/cpp/language/throw dla C++
- https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html dla Javy
- https://docs.python.org/2/tutorial/errors.html dla Pythona

Styl programowania

Dobrze jest przestrzegać reguł związanych ze stylem programowania. Można w tym celu wykorzystać zasady zamieszczone w artykule:

http://geosoft.no/development/cppstyle.html

przede wszystkim ważne są:

- spójność nazewnictwa zmiennych i typów,
- spójność w zakresie stosowania tabulacji (wcięcia) i odstępów,
- ograniczony rozmiar funkcji,
- zachowanie spójności w organizacji kodu źródłowego wewnątrz klasy (np. jednolita kolejność public->protected->private).