## Горизонтальная линия



Semestrální práce

z předmětů testování softwaru

projekt pro testování: GameEngine

Ivan Shalaev

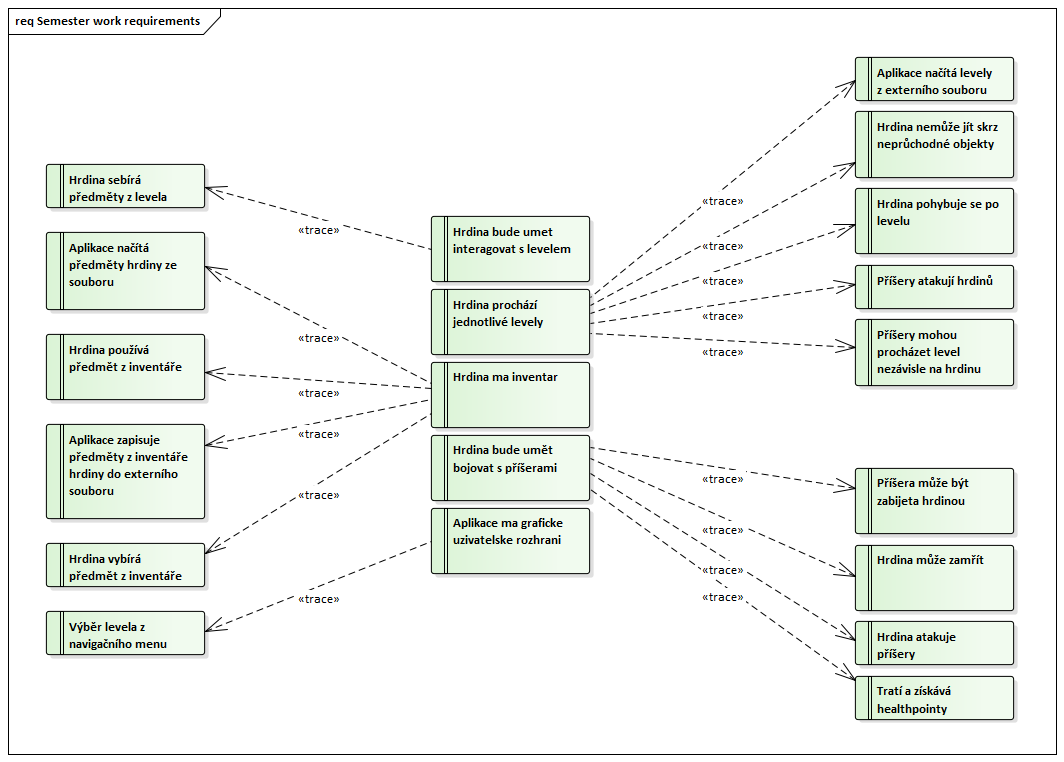
FEL ČVUT 2021

[shalaiva@fel.cvut.cz](mailto:shalaiva@fel.cvut.cz)

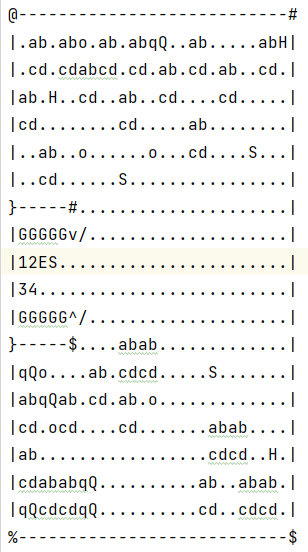
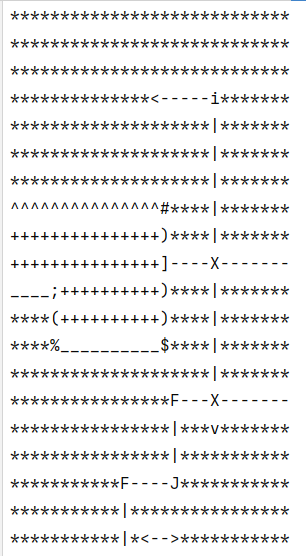
# Návrh testovací strategie

## Popis projektu pro testování

Projektem je jednoduchý herní engine, který jsem vytvořil v rámcích semestrální práce z předmětu PJV. Ze specifikací požadavku semestralky z PJV můžeme vyjádřit systémové požadavky projektu rozdělením původních požadavků na dílčí části.

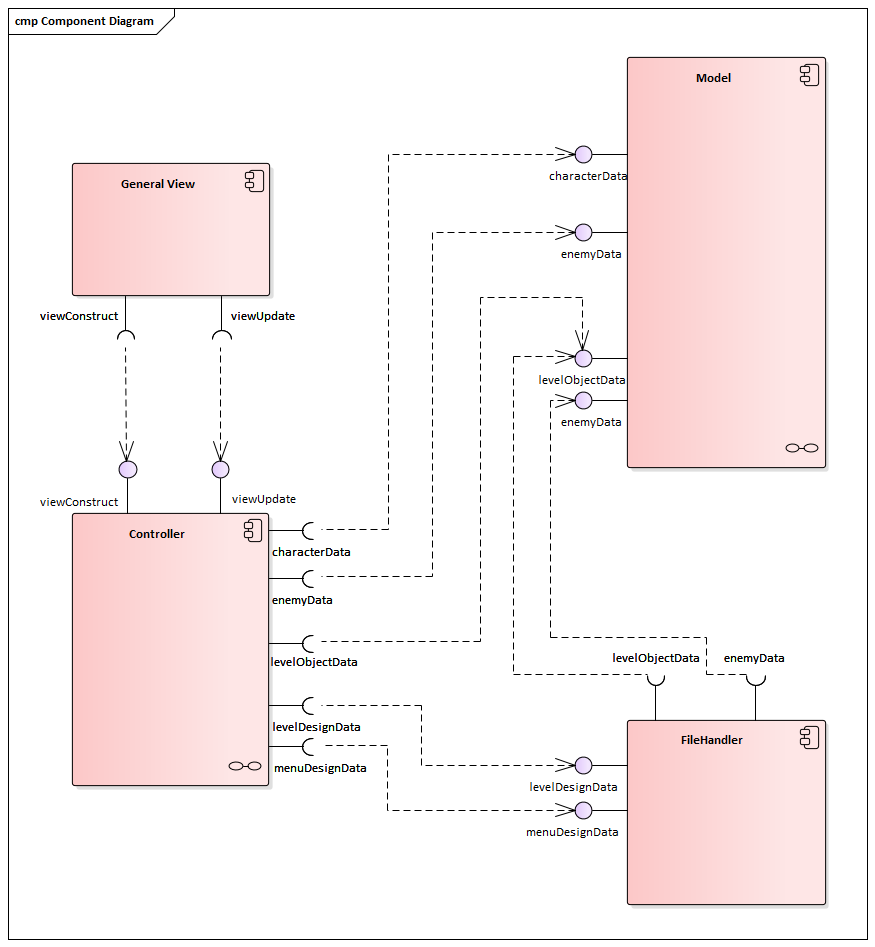






Navíc jsem ještě implementoval jednoduchý generátor objektů na úrovni z textovou mapy. Dále jsou screenshoty mapy objektu a tilu zemi

## Testovaci strategie

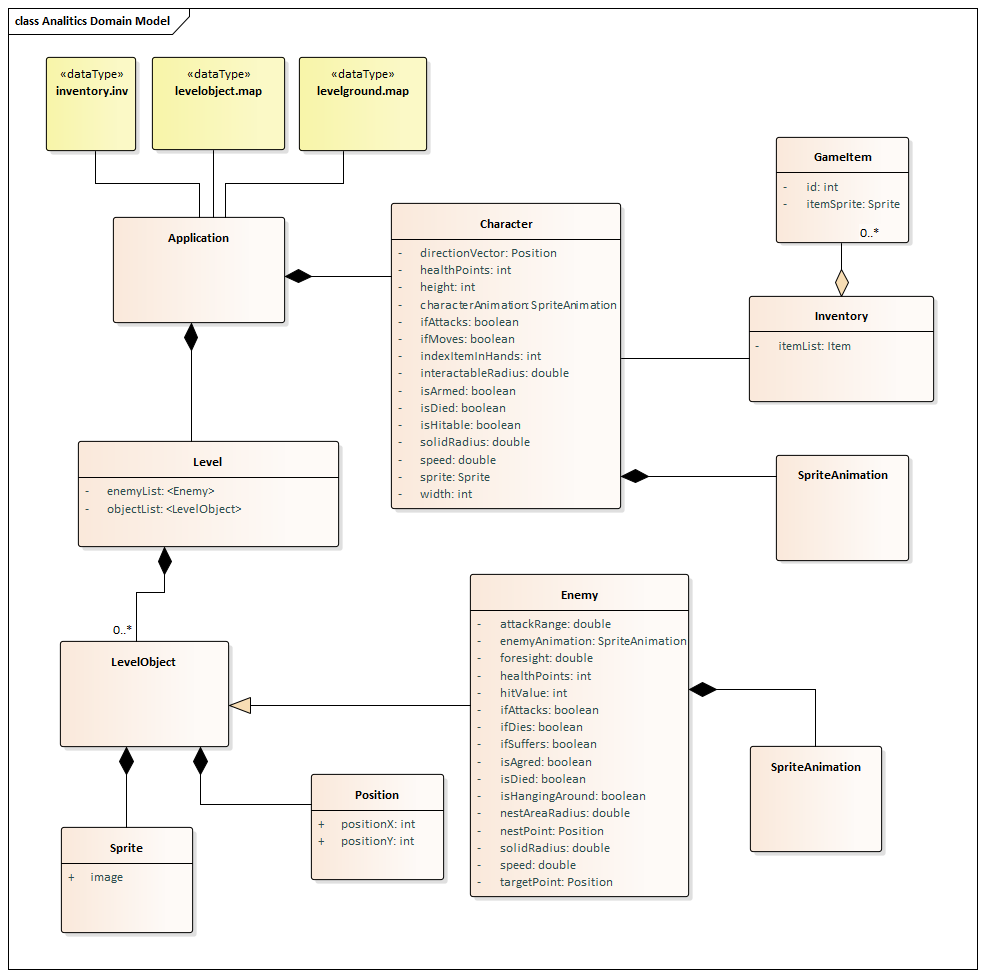
Samotný program byl sestaven podle architektury MVC a podle ní bych rozděloval celý program na komponenty.

Tím pádem máme **části**:

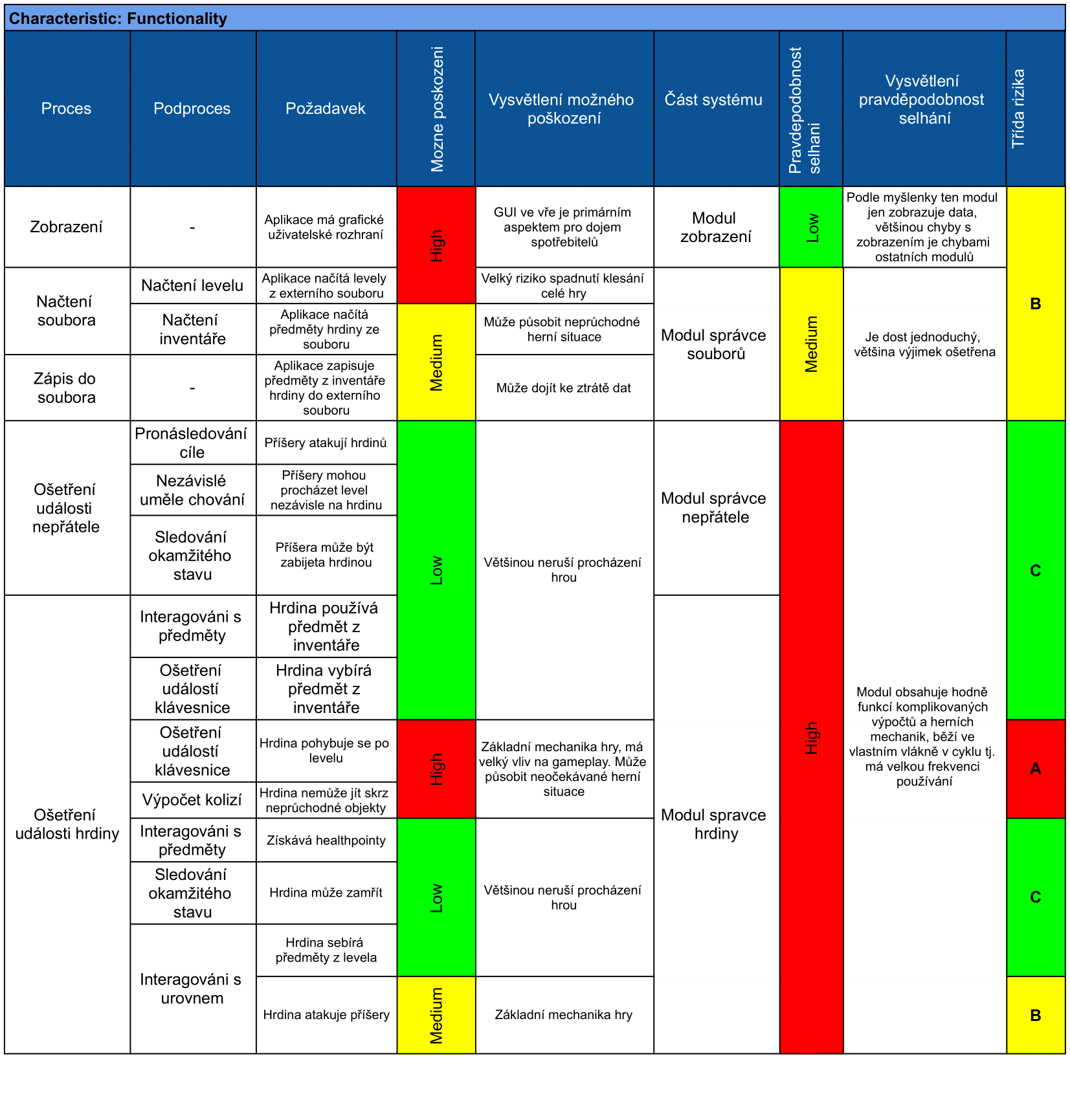
* Modul správce úrovně
* Modul správce hrdiny
* Modul správce nepřátele
* Modul zobrazení
* Modul správce souborů

A **procesy**:

* Načtení soubora
* Zápis do soubora
* Zobrazení (grafické)
* Ošetření události hrdiny
* Ošetření události nepřátele



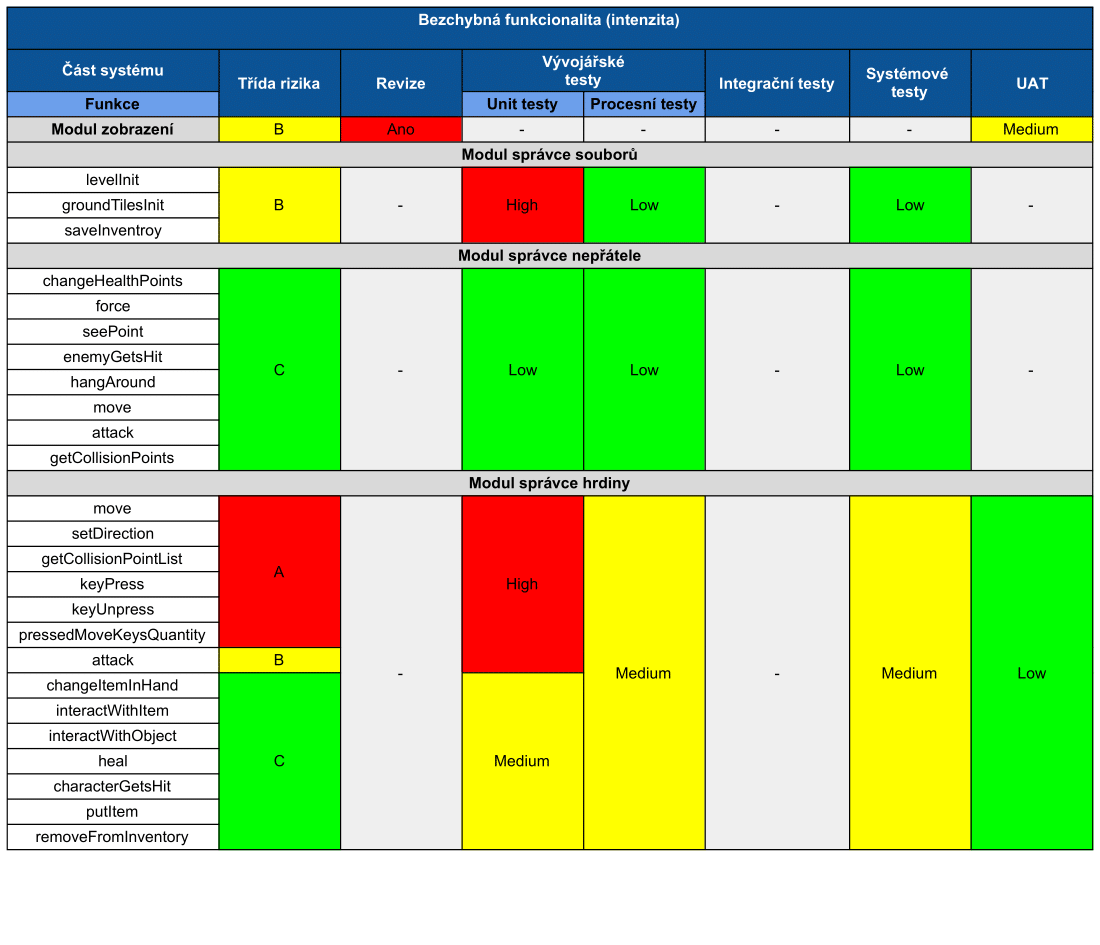
**Doplněná tabulka tříd rizika**

****

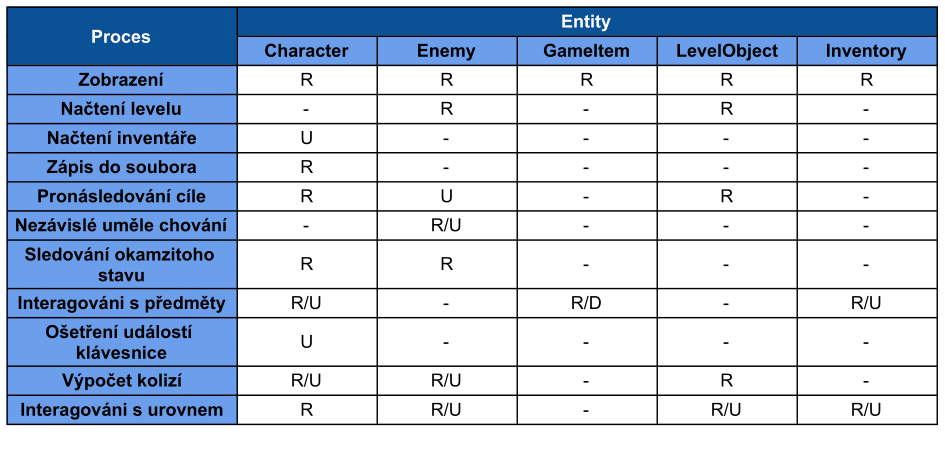
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Pravděpodobnost selhání** | | |
| **High** | **Medium** | **Low** |
| **Možné poškození v případě selhání** | **High** | A | B | B |
| **Medium** | B | B | C |
| **Low** | C | C | C |

### **Určení intenzity**

* V rámcích této semestralky budeme uvažovat Unit testy a Procesní testy. V projektu GameEngine nejsou poskytnuté souvislostí s jinými systémy a proto, v souladu s definicí, Integrační testy provázet nebudeme.

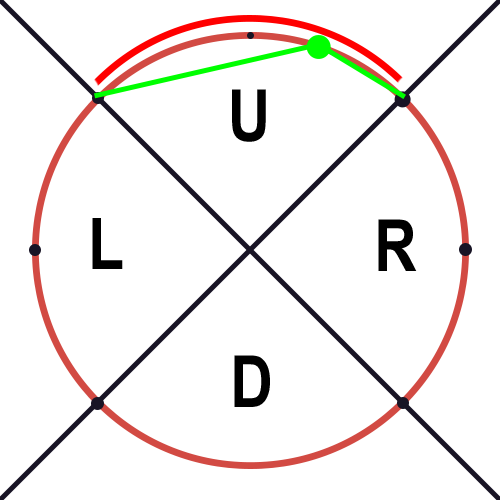


**CRUD Matice**

****

## Testovací scénáře

### **Třídy ekvivalence pro funkci setDirection \*EnemyController\***

* S dokumentací víme: tato funkce vrací prostou stringovou reprezentaci směrového vektoru nepřítele. Tato funkce počítá délky úseček od konce jednotkového směrového vektoru do dvou diagonálních bodu (např. do bodu UP-LEFT a UP-RIGHT), pak skládá ty hodnoty a porovnává s délkou ¼ kružnice. Když ten součet je menší, zapisuje do řetězce slovní reprezentaci směru (napr. ÚP).

Tříd ekvivalence bude 4 podle výstupů a každá bude reprezentována intervalem vektoru mezi diagonálními body V dokumentaci není jednoznačně určený mezní podmínky a proto řekneme, že vektory s končí v UP-LEFT a DOWN-LEFT patří do EC “LEFT” a vektory s konci UP-RIGHT a DOWN-RIGHT patří do EC “RIGHT”. Jedná se o jednotkových vektorech a o kružnici s r = 1. Z toho určíme vektory, generující každou EC (A = , v = (jednotkový) vektor s konci v UP-RIGHT) :

1. LEFT =
2. RIGHT =
3. UP =
4. DOWN =

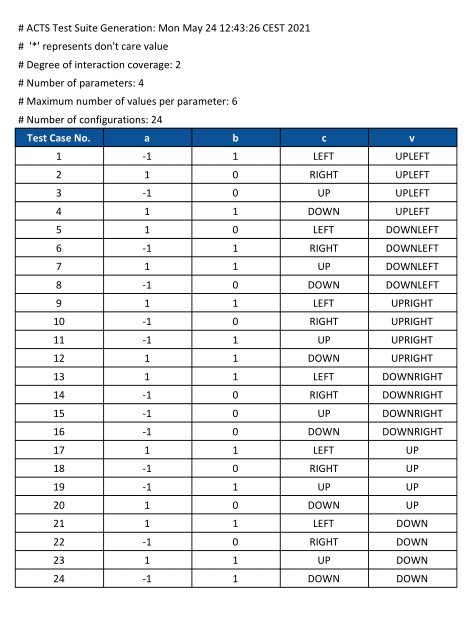
V programu pro reprezentaci vektorů se používá třída Position mající parametry x a y. Z toho máme:

1. LEFT =
2. RIGHT =
3. UP =
4. DOWN =

Vím, že tato funkce má triviální vstup, proto uměle zkomplikujeme vstupy a řekněme, že kromě směrového vektoru funkce bude přijímat na vstup jese 3 parametry: int a, int b, String č. Doplňková podmínka je, když a > b výstup je c, jinak podle logiky funkci výše. Z toho určíme nové EC:

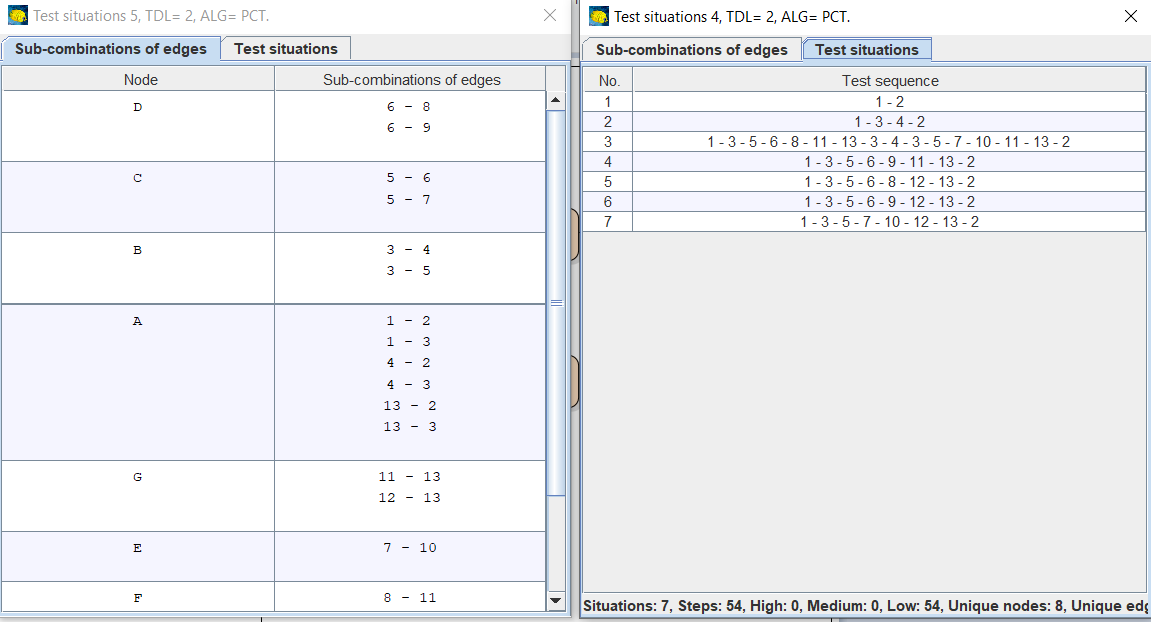
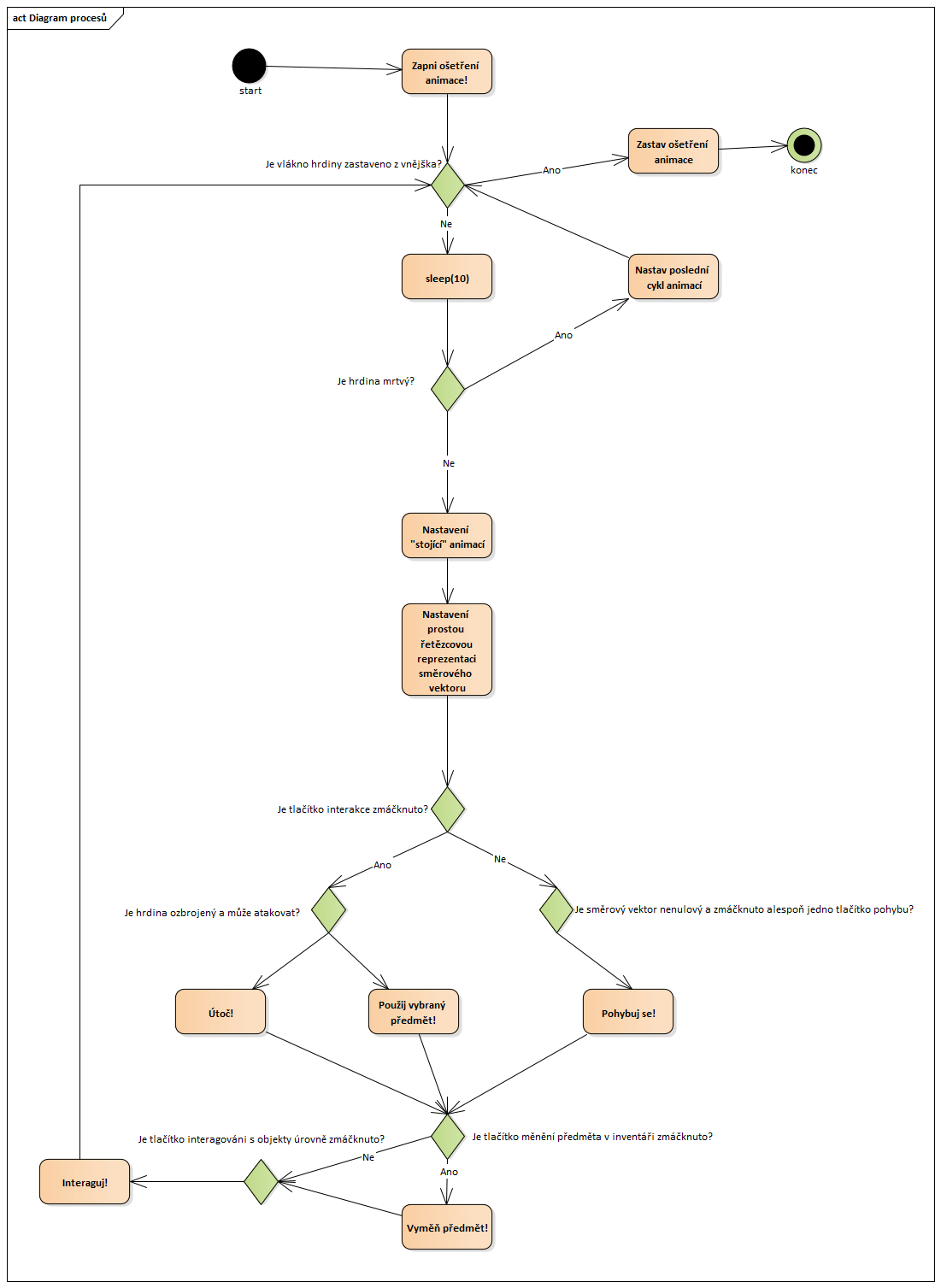
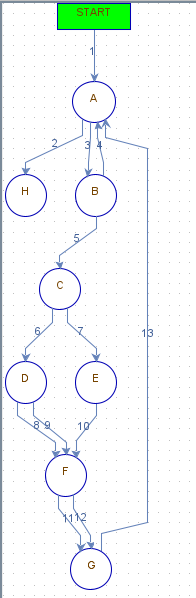
1. LEFT =
2. RIGHT =
3. UP =
4. DOWN =

Určíme vstupní data pokrývající všechny EC:

* ,
* ,
* ,, ,
* , 
* ,

Z toho máme scénáře kde pro ***v*** platí:

### **Procesní diagram (ošetření události hrdniny)**



### **Detailní testovací scénář**

|  |  |
| --- | --- |
| **Parametr** | **Obsah** |
| ID testu | 4 |
| Sekvence | 1-3-5-6-9-11-13-2 |
| Název testu | Test ošetření události hrdiny |
| Hloubka detailů | Střední |
| Popis testu | Hrdina pohybuje se po úrovni, zmačkńe tlačítko interagování s předmětem, interagování s urovnem, interagování s inventářem a zemře |
| Vstupní podmínky | Živý hrdina má v inventáři předmět (kromě zbroje) |
| Testovací data | * character.inventory * simpleDir |
| Očekávaný výsledek | Prázdný inventář, simpleDir=”RIGHT” |

## Přehled implementací testů

[InventoryFileHandlerTest](https://gitlab.fel.cvut.cz/shalaiva/ts1-semester-work/-/blob/master/GameEngine/src/main/test/cz/cvut/fel/pjv/InventoryFileHandlerTest.java)

Test funkci saveInventory, prověřuje správnost zápisu inventáře do souboru (podle doku přepisuje se IDčka v soubor save.inv), používáno mockování pro třídu Inventary.

[CharacterControllerTest](https://gitlab.fel.cvut.cz/shalaiva/ts1-semester-work/-/blob/master/GameEngine/src/main/test/cz/cvut/fel/pjv/CharacterControllerTest.java)

3 unit testy s mockováním:

**move\_directionVectorUP\_movesUp()** - mockuje se *directionVector* (hrdina posune se ve směru *directionVector*), spustí se *move()*, ověřuje se posun hrdiny podle ho posize.

**changeItemInHand\_2items\_2ndItemInHand()** - mockuje se *Inventory*, “vkládá se” do toho mocky *Item* a *Weapon*(dědí se od *Item*), vytváří se hrdina s tímto inventářem (defaultně první předmět je *itemInHand*), spustí se *changeItemInHand()*, ověřuje se, že *itemInHand* je ten *Weapon*.

**interactWithItem\_2items\_2ndItemInHand()** - mockuje se *Inventory*, vkládá se” do toho mocky *Item*’y, přičemž jeden předmět je heal, tj. interaktivní (podle IDčka), vytváří se hrdina s tímto inventářem (defaultně první předmět je *itemInHand*), spustí se *interactWithItem()*, ověřuje se, že *itemInHand* je teď ten druhý předmět (první byl použit a smazán)

4 procesni testy:

**processTest1()** - testuje kombinace metod *setDirection()*, *move()*, *characterGetsHit()*, *interactWithItem()*, po každé metodě se ověřuje správnost výsledků:

*setDirection()* mění *simpleDir*

*move()* mění pozice

*characterGetsHit()* mění HP hrdiny

*interactWithItem()* v tomto případě mění HP (používá se heal)

**processTest2()** - testuje kombinace metod *changeItemInHand()*, *attack()*, *characterGetsHit()*, po každé metodě se ověřuje správnost výsledků:

*changeItemInHand()* - mění *itemInHand*

*attack()* - mění *attacks* (bool proměnná)

*characterGetsHit()* - v tomto případě mění *isDied* (bool proměnná)

**procesniTest3()** - testuje kombinace metod *changeItemInHand()*, *act()*, *changeItemInHand()* x3, *interactWithItem()*, po každé metodě se oveřuje správnost výsledků:

*changeItemInHand()* - mění *itemInHand*

*act()* - mění *acts* (bool proměnná)

*interactWithItem()* - mění *itemInHand*

**procesniTest4()** - testuje kombinace metod *keyPress()*, *setDirection()*, *move()*, *keyUnpress()*, *setDirection()*, *move()*, po každé metodě se ověřuje správnost výsledků:

*keyPress()* - nemůžeme ověřit vliv metody, proto ověříme nepřímo po volání následující metody

*setDirection()* - mění *simpleDir* (v závisnosti od *keyPressed()* )

*move()* mění pozici

[PositionTest](https://gitlab.fel.cvut.cz/shalaiva/ts1-semester-work/-/blob/master/GameEngine/src/main/test/cz/cvut/fel/pjv/Model/PositionTest.java)

Parametrické unit testy funkcí třídy *Position*. Všechny funkci *Position* mají nekonečně mnoho EC (podle výstupů), proto jsem napsal parametrické testy pokrývající celé/desítkové a kladné/záporné vstupy