Dokumentáció a projekt laborítórium LaTeX sablonjaihoz.

2024. február 27.

A következő fejezetekben a *definitions* fájlban található környezet és parancs definíciók kerülnek leírásra. Ahol beépített LaTeXelemet említek külön jelzem.

1. Általános környezetek és parancsok

Egyszerű képek beszúrására alkalmas a *diagram* parancs, ami számozza a beszúrt képeket és mindig középre igazítja. 3 paraméterrel rendelkezik:

1. A kép elérési útja és neve, kiterjesztést nem kell megadni. Az elérésiút, mindig a fő dokumentum mappájából kiindulva adható meg.
2. A képhez tartozó leírás megadására szolgál.
3. A kép átméretezésére szolgáló paraméter: cm/px/in... mértékegységgel megadva.

Egy példaparancs és eredménye:

```
\diagram{../docs/img/BMElogo}{Példa kép}{3cm}
```

1. ábra. Példa kép

A dokumentumban számozott definíciók és rövidítések megadására szolgálnak a következő környezetek, példaprogrammal:

```
\begin{defi}
    Első definíció ...
\end{defi}
\begin{rov}
    Első rövidítés
\end{rov}
```

1. Definíció. *Első definíció ...*

1. Rövidítés. *Első rövidítés*

2. Napló használata

Minden egyes beadandónál le kell adni, az abban a szakaszban elkészített dokumentációkról és programról egy naplót. Ezt a *naplo* környezet segítségével adhatjuk meg, a napló bejegyzéseit a *naplotag* parancs szimbolizálja, mely 4 paramétert vár a napló fejlécének megfelelő sorredben, azaz a feladat elkezdésének időpontja, a feladat időtartama, a feladatban résztvevők, valamint leírás megadása. A leírás oszlopban, azaz az utolsó paraméterben lehetőségünk van a sorok tördelésére a *newline* segítségével.

```
\begin{naplo}
    \naplotag{feb. 18. 16h }{ 1 óra }{ Csapat }{ Értekezlet a tagok között \newline
    Döntés: Segédeszközök kiválasztása (git, trello, drive)}
    \naplotag{feb. 19. 16h }{ 1.5 óra }{ Teszt János }{ Funkciók megtervezése}
\end{naplo}
```

Kezdet	Időtartam	Résztvevők	Leírás
feb. 18. 16h	1 óra	Csapat	Értekezlet a tagok között Döntés: Segédeszközök kiválasztása (git, trello, drive)
feb. 19. 16h	1.5 óra	Teszt János	Funkciók megtervezése

3. Követelménydokumentálás

Általános követelménynek leírására a *kovetelmeny* környezet szolgál, megadási módja: A környezet első paramétere a követelmény azonosítója, majd sorrendben a piorítás, forrás, ellenőrzés és leírás. Ezeket minden esetben meg kell adni paraméterként, azaz {} között. Opcionálisan a paraméterek után szabadon megadhatunk megjegyzést.

```
\begin{kovetelmeny}
    {R31}
    {Alapvető}
    {Feladatkiadó}
    {bemutatás}
    {A játék futtatásához tetszőleges operációs rendszer alkalmas, amin elérhet...}
    Természetesen elvárás, hogy olyan illesztőprogramok telepítve legyenek, amik ...
\end{kovetelmeny}
```

Azonosító	Prioritás	Forrás	Ellenőrzés
R31	Alapvető	Feladatkiadó	bemutatás

Leírás: A játék futtatásához tetszőleges operációs rendszer alkalmas, amin elérhető a Java Runtime Environment (JRE, legalább JRE 8-as verzió) , más szoftver nem szükséges.
Természetesen elvárás, hogy olyan illesztőprogramok telepítve legyenek, amik lehetővé teszik azon perifériák kezelését, amik a játékhoz kellenek (R33).

Funkcionális követelményekhez a *funkovetelmeny* környezet való, használata részben megegyezik az előző környezettel. Rendre a következő paraméterek megadása szükséges: azonosító, pioritás, forrás, use-case neve, ellenőrzés, leírás és opcionálisan megjegyzés.

```
\begin{funkovetelmeny}
    {R26}
    {Fontos}
    {Feladatkiírás}
    {Játék megtekintése, Jelzőrakéta építés}
    {Bemutatás}
    {Egy egységnyi munkával összeszerelhető és elsüthető a jelzőrakéta, ami...}
\end{funkovetelmeny}
```

Azonosító	Prioritás	Forrás	Use-case	Ellenőrzés
R26	Fontos	Feladatkiírás	Játék megtekintése, Jelzőrakéta építés	Bemutatás

Leírás: Egy egységnyi munkával összeszerelhető és elsüthető a jelzőrakéta, ami a játék megnyerését jelenti.

4. Használati esetek, use-case-ek megadása

Használati eset leírására a *use – case* környezet használható, mely sorrendben a következő paramétereket várja: használati eset neve, rövid leírása és a use-case-ben résztvevő aktorok. A környezet főtörzsébe a forgatókönyv tetszőleges formázással megadható. A forgatókönyvnél az alternatívákat egy új sor beszúrásával, a korábban említett *newline* paranccsal valamint kiemeléssel (*textbf* parancs) szemléltetjük.

```
\begin{use-case}
    {Képesség használat}
    {A játékos a szereplő specifikus képességeit használhatja.}
    {Player}
    A játékos kijátssza a képességet. \newline
       \textbf{1.A.1} A sarkkutató játékos megnézheti, hogy egy adott szomszédos vagy a
       saját mezőjén hány játékos állhat. \newline
       \textbf{1.B.1} Az eszkimó játékos iglut építhet a kiválasztott mezőre.
\end{use-case}
```

Use-case neve:	Képesség használat
Rövid leírás:	A játékos a szereplő specifikus képességeit használhatja.
Aktorok:	Player
Forgatókönyv:	A játékos kijátssza a képességet. **1.A.1** A sarkkutató játékos megnézheti, hogy egy adott szomszédos vagy a saját mezőjén hány játékos állhat. **1.B.1** Az eszkimó játékos iglut építhet a kiválasztott mezőre.

5. Osztályok, interfészek dokumentálása

Az osztály vagy interfész elemeinek megadására különböző környezeteket használhatunk:

- *class – template – responsibility* Az osztály felelősségének leírására szolgáló környezet
- *class – template – baseclass* Az ősosztályok felsorolására használható, az öröklés reprezentálása a *baseclass* paranccsal történik.
- *class – template – interface* A megvalósított interfészek felsorolására
- *class – template – association* Az osztályhoz tartozó kapcsolatok/asszociációk felsorolása és leírása. A felsoroláshoz a *classitem* parancs használható, mely két paramétert vár: az adattag típusa, neve, megkötések... valamint a hozzá tartozó leírást
- *class – template – attribute* Az osztályban található attribútumok felsorolása és leírása. A felsoroláshoz a *classitem* parancs használható, mely két paramétert vár: az adattag típusa, neve, megkötések... valamint a hozzá tartozó leírást
- *class – template – method* Az osztályban megvalósuló függvények leírása, felsoroláshoz itt is a *classitem* parancs kell.
- *class – template – statechart* Ha szükséges, állapotgépek leírásához.

Egy példa néhány osztálykörnyezet és a *classitem* parancs használatára

```
\subsection*{DivingSuit}
\begin{class-template-responsibility}
    Absztrakt búvárruha osztály.
\end{class-template-responsibility}
\begin{class-template-baseclass}
    Item \baseclass AutomaticItem
\end{class-template-baseclass}
\begin{class-template-association}
    \classitem{-player: Player}{A tárgyat birtokló játékosra vonatkozó hivatkozás.}
\end{class-template-association}
\begin{class-template-method}
    \classitem{+fallInWater(p: Player): void}{Absztrakt, felülírható metódus, a ... }
\end{class-template-method}
```

DivingSuit

■ Felelősség

Absztrakt búvárruha osztály.

■ Ősosztály

Item → AutomaticItem

■ Asszociációk

◇ *-player: Player* - A tárgyat birtokló játékosra vonatkozó hivatkozás.

■ Metódusok

◇ *+fallInWater(p: Player): void* - Absztrakt, felülírható metódus, a játékos vízbe esésénél van szerepe.

Lehetőség van felsorolás típus leírására is a *enum – template* környezet segítségével, mely két paramétert vár: leírás és a literálok felsorolása.

6. Tesztek leírása

Tesztek összefoglaló leírására a *test* parancs használható, mely a következő paramétereket várja: a teszt-eset neve, a teszt rövid leírása és a teszt célja.

```
\test{A játék szimulálás}{A start automated parancs tesztesete}
{A parancs hatására a játék elindul, az egyes aktorok lépnek, léphetnek.}
```

Teszt-eset neve	A JÁTÉK SZIMULÁLÁS
Rövid leírás	A start automated parancs tesztesete
Teszt célja	A parancs hatására a játék elindul, az egyes aktorok lépnek, léphetnek.

A tesztek részletes leírására a következő környezetek használhatók:

- *test – case – description* a teszt leírása
- *test – case – function* ellenőrzött funkciók, várható hibák
- *test – case – input* a teszthez tartozó bement
- *test – case – output* a teszt elvárt kimenete és eredménye

A teszt bemenetének és elvárt kimenetének megadására használandó a beépített *verbatim* környezet, ami a szöveget mindenféle formázás nélkül, úgy jeleníti meg, ahogy beleírták (a tabokat nem jeleníti meg, helyette szóköz javasolt).

```
\subsection*{A játékvilág létrehozása}
\begin{test-case-description}
    A teszteset célja az \texttt{init world} parancs lefutásának ellenőrzése.
\end{test-case-description}
\begin{test-case-function}
    A teszt alatt leellenőrizzük, hogy ...
\end{test-case-function}
\begin{test-case-input}
   \begin{verbatim}
        init world
        0: S 3
        ...
        print
    \end{verbatim}
\end{test-case-input}
\begin{test-case-output}
    A \texttt{print} parancs hatására a következő kell megjelenjen a kimeneten:
    \begin{verbatim}
        worlddata
        ...
        creatures 0
    \end{verbatim}
\end{test-case-output}
```

A játékvilág létrehozása

■ Leírás

A teszteset célja az `init world` parancs lefutásának ellenőrzése.

■ Ellenőrzött funkcionalitás, várható hibahelyek

A teszt alatt leellenőrizzük, hogy ...

■ Bemenet

```
init world
0: S 3
...
print
```

■ Elvárt kimenet

A `print` parancs hatására a következő kell megjelenjen a kimeneten:

```
worlddata
...
creatures 0
```

A teszteket eredményük szempontjából egyszerűen két csoportba oszthatók: a teszt sikeresen lefutott, vagy valamin elhasalt (most lényegtelen, hogy ezt mi okozta). Előbbi esetén a *testOK* parancs míg utóbbi esetén a *testFAIL* parancs használandó. Helyes teszt esetén a két paraméter a tesztelő neve és a teszt időpontja, míg rossz teszteset esetén ez a teszt hibás eredményével, a hibaokok és a változtatások leírásával bővül a paraméterlista.

```
\subsection*{Ásás törékeny ásóval}
\testFAIL{LaTex}{Ápr. 25. 13:50}{A törött ásó hátralevő használatai ...}
    {Akkor is csökkenti a BrokenShovel use-ja a use\_num-ját, ha már el van törve.}
    {Csak akkor csökken a use\_num, ha nagyobb, mint 0.}
\testOK{LaTex}{Ápr. 25. 14:00}
```

Ásás törékeny ásóval

Tesztelő neve	LaTex
Teszt időpontja	Ápr. 25. 13:50
Teszt eredménye	A törött ásó hátralevő használatai negatívba megy, ha próbáljuk használni, miután eltörött.
Lehetséges hibaokok	Akkor is csökkenti a BrokenShovel use-ja a use_num-ját, ha már el van törve.
Változtatások	Csak akkor csökken a use_num, ha nagyobb, mint 0.

Tesztelő neve	LaTex
Teszt időpontja	Ápr. 25. 14:00

7. Adminisztratív információk megjelenítése

Ütemtervek vagy határidőnapló beszúrására a *terv* környezet és a *tervitem* parancs együtt használandó. A meglenített táblázatban a következők szerepelnek: feladat határideje, a feladat leírása és a szakasz felelőse, rendre ezen paramétereket várja a *tervitem* parancs is:

```
\begin{terv}
    \tervitem{ápr. 4}{Grafika 1. hf leadása}{ ?????? }
    \tervitem{ápr. 25}{Grafika 2. hf leadása}{ ?????? }
\end{terv}
```

Határidő	Feladat	Felelős
ápr. 4	Grafika 1. hf leadása	??????
ápr. 25	Grafika 2. hf leadása	??????

Szótárak, vagy kulcs-érték párok felsorolására a *szotar* környezet és *szotaritem* parancs együtt használandó. A parancs 2 paramétert vár: kulcs és érték.

```
\begin{szotar}
    \szotaritem{Kulcs 1 }{Érték 1}
    \szotaritem{Hóásás }{Egy réteg hó eltakarítása, azaz törlése az adott...}
\end{szotar}
```

Kulcs 1 Érték 1

Hóásás Egy réteg hó eltakarítása, azaz törlése az adott jégtábláról. Egy egység munkába kerül.

Fájlok beadása esetén a *fajllista* környezet és *fajl* parancs használandó. A parancsnak rendre a következő 4 paramétere van: fájl neve, mérete, keletkezés ideje, leírás.

```
\begin{fajllista}
    \fajl{AutomaticItem.java}   {353 byte} {2020.03.26~21:05} {AutomaticItem osztály}
    \fajl{BasicDivingSuit.java} {797 byte} {2020.03.26~20:12} {BasicDivingSuit osztály}
\end{fajllista}
```

Fájl neve	Méret	Keletkezés ideje	Tartalom
AutomaticItem.java	353 byte	2020.03.26 21:05	AutomaticItem osztály
BasicDivingSuit.java	797 byte	2020.03.26 20:12	BasicDivingSuit osztály

A szakaszonkénti teljesítményértékelések megadásához az *ertekeles* környezet és *ertekelestag* parancs használandó:

```
\begin{ertekeles}
    \ertekelestag{LaTex Zoltán}{??????}{20\%}
    \ertekelestag{Git Attila}{??????}{20\%}
\end{ertekeles}
```

Tag neve	Tag neptun	Munka százalékban
LaTex Zoltán	??????	20%
Git Attila	??????	20%

A végső beadásnál az elvégzett munka összefoglalására az *ertekelesOra* környezet használandó az előzőleg már alkalmazott *ertekelestag* parancs felhasználásával:

```
\begin{ertekelesOra}
    \ertekelestag{Teszt Tamás}{??????}{41}
    \ertekelestag{Java Dániel}{??????}{40}
\end{ertekelesOra}
```

A végső beadásnál az beadott munkák összefoglalására az *ertekelesKod* környezet használandó az *ertekelestagk* parancs felhasználásával:

Tag neve	Tag neptun	Munka órában
Teszt Tamás	??????	41
Java Dániel	??????	40

```
\begin{ertekelesKod}
    \ertekelestagk{Szkeleton}{853}
    \ertekelestagk{Prototípus}{1357}
    \ertekelestagk{Grafikus változat}{2605}
    \ertekelestagk{Összesen}{4815}
\end{ertekelesKod}
```

Fázis	Kódsorok száma
Szkeleton	853
Prototípus	1357
Grafikus változat	2605
Összesen	4815

8. További hasznos parancsok

- `\clearpage` egy új oldal megkezdése (beépített parancs)
- `\urlref{url cím}{leírás}` hivatkozás beszúrása
- `\newline` új sor kezdése (beépített parancs)
- `\textbf{szoveg}` félkövér szöveg (beépített parancs)
- `\textit{szove}` dölt betűk (beépített parancs)
- `\comment{szoveg}` piros, jobbra igazított szöveg beszúrása
- `\setcounter{chapter}{-1}` A fejezetek számozásának átállítására (mindig a megadott számtól folytatódik a számozás, itt 0-tól) (beépített parancs)
- `\setcounter{section}{-1}` A szakaszok számozásának átállítására (mindig a megadott számtól folytatódik a számozás, itt 0-tól) (beépített parancs)
- `\tableofcontents` Tartalomjegyzék megjelenítése a végső beadásnál (beépített parancs)
- `\input{fájl}` Egyes szakaszok nagyon nagy méreteket ölthetnek, így érdemes külön fájlba rakni, ennek behúzására alkalmas a parancs. (beépített parancs)

9. Scriptek

9.1. Szótár generálás fájlból

A *szotar.sh* bash script alkalmas egy szöveges fájlból, amiben a kulcs-érték párok külön-külön sorban vannak és egy speciális karakterrel el vannak választva, abc-sorrendbe rakni és a már említett szintaktikát létrehozni a *szotar* környezet számára. Vagyis a konzolon megjelent szöveget a `\begin{szotar} \end{szotar}` közé kell beilleszteni, ugyanakkor érdemes külső fájlba rakni, és az *input* parancsot használni.

Alapértelmezetten a # karakter mentén történik a szétválasztás, de a 2. paraméterben megadhatunk eltérő karaktert is, így excel kompatibilis is lehet. Windows esetén PowerShell-el tudjuk a programot futtatni. Példák:

```
./szotar.sh szotar.txt
./szotar.sh szotar.txt "-"
./szotar.sh szotar.txt "|" > out.txt
```

9.2. Fájllista generálása

Az aktuális mappában kiadva következő parancsot, a konzolra kiíródik a *fajllista* környezetbe beillesztendő formázott lista:

```
stat -c "\fajl{%n}{%s byte}{%.16y}{...}" *
```

Az output átirányítása fájlba:

```
stat -c "\fajl{%n}{%s byte}{%.16y}{...}" * > fajllista.txt
```

Sajnos egyes rendszereken a $POSIX$ szabvány miatt, nem a támogatja a fájlrendszer a létrehozási időpont tárolását. Ezért itt a $\%.16y$ az utólsó módosítást adja meg, ha a rendszer támogatja a létrehozási időpontot, akkor a $\%.16w$ paraméterrel lehetne lekérdezni.