Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Diplomová práce

Otázkový modul/y pro testování simplexového algoritmu do LMS Moodle

Bc. Milan Jelínek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Milan Jelínek

Systémové inženýrství a informatika
Informatika

Název práce

Otázkový modul/y pro testování simplexového algoritmu do LMS Moodle

Název anglicky

Question type module/s for simplex algorithm testing for LMS Moodle

---

Cíle práce

Diplomová práce se zabývá analýzou, návrhem a implementací otázkového modulu/ů pro zrychlení a zjednodušení zadávání úloh na simplexový algoritmus do LMS Moodle. Tento modul/y bude/ou řešit otázky otevřené, uzavřené a uzavřené s možnostmi.

Metodika

Diplomová práce se sestává ze dvou částí.Metodika zpracování teoretické části práce je založena na studiu odborných a vědeckých informačních zdrojů. Na základě syntézy zjištěných požadavků budou formulována teoretická východiska pro zpracování praktické části práce.

Praktická část práce spočívá v návrhu a implementaci modulu/ů. Při návrhu a implementaci bude využito standardních metod a nástrojů softwarového inženýrství. Knihovna bude prakticky otestována a budou shrnuty zásadní poznatky zjištěné při její implementaci a testování a budou navrženy možnosti případného dalšího budoucího rozšíření.

Doporučený rozsah práce
80

Klíčová slova
Moodle, question type, testy, operační výzkum, PHP, simplexový algoritmus

---

Doporučené zdroje informací
Doug Bierer: PHP 7 Programming Cookbook PHP, Packt, ISBN-10: 1785883445
Padraic Brady: Survive the Deep End: PHP Security, ReadTheDocs.org
Paul Hudson: PHP in a Nutshell, O'Reilly Media, ISBN-10: 0-596-10067-1
Phil Sturgeon, Josh Lockhart: PHP: The Right Way, PhpTheRightWay.com
Steven S. Skiena: The Algorithm Design Manual -Second Edition, Springer, ISBN-10: 1849967202

Předběžný termín obhajoby
2021/22 LS – PEF

Vedoucí práce
prof. RNDr. Helena Brožová, CSc.

Garantující pracoviště
Katedra systémového inženýrství

Konzultant
Ing. Dana Vynikarová, Ph.D.

Elektronicky schváleno dne 24. 11. 2021
doc. Ing. Tomáš Šubrt, Ph.D.
Vedoucí katedry

Elektronicky schváleno dne 25. 11. 2021
Ing. Martin Pelikán, Ph.D.
Děkan

V Praze dne 01. 02. 2022

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Otázkový modul/y pro testování simplexového algoritmu do LMS Moodle" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2022 _____

Poděkování

Rád bych touto cestou poděkoval prof. RNDr. Heleně Brožové, CSc. za vedení mé diplomové práce, poskytnuté rady, a konzultace.

Také bych chtěl poděkovat své rodině, za veškerou podporu při psaní diplomové práce a během celé doby studia.

Otázkový modul/y pro testování simplexového algoritmu do LMS Moodle

Abstrakt

Diplomová práce se zabývá analýzou problematiky testování simplexového algoritmu. Na základě poznatků analýzy je zvoleno místo pro časovou optimalizaci testování. V další části se práce zabývá návrhem takové optimalizace a implementaci řešení v LMS Moodle. Implementované řešení umožnuje zadaní parametrů simplexového algoritmu v kanonickém tvaru. Systém následně úlohu vyřeší a umožní zadavateli úlohy vytvořit testovou úlohu pro studenta z vybraných hodnot postupu výpočtu či z hodnot nejlepšího řešení pro danou úlohu.

Klíčová slova: LMS Moodle, PHP, Formulas, Simplexový algoritmus, Testování

Question type module/s for simplex algorithm testing for LMS Moodle

Abstract

This diploma thesis focuses on analysis of the simplex algorithm testing. As revealed by the analysis, a place for time optimization of testing is chosen. The following section of the thesis deals with design of said optimization and implementation of the solutions in Moodle LMS. The solution allows entry of parameters of the simplex algorithm in canonical form. The system then solves the problem and permits assigner to create a test question for a student from selected values from the calculation procedure or the values for the best solution to the task.

Keywords: LMS Moodle, PHP, Formulas, Simplex algorithm, Testing

Obsah

1 Úvod **11**

2 Cíl práce a metodika **12**

2.1 Cíl práce 12

2.2 Metodika 12

3 Teoretická východiska **13**

3.1 Současný stav testování 13

3.1.1 Úvod do testování 13

3.1.2 Testování simplexového algoritmu 14

3.1.3 Testování simplexového algoritmu v LMS Moodle 15

3.2 LMS Moodle 16

3.2.1 Moodle struktura 16

3.2.2 Otázka 18

3.2.3 Question bank 22

3.2.4 Question engine 22

3.2.5 Quiz 23

3.2.6 Moodle question type 24

3.2.7 Kategorie testových úloh 25

3.2.8 Štítky 26

3.2.9 Formulas 27

3.3 PHP 30

3.4 Operační analýza 31

3.4.1 Lineární programování 32

3.4.2 Postup řešení úlohy lineárního programování 32

3.4.3 Kanonický tvar v lineárním programování 35

3.5 Simplexový algoritmus 36

3.5.1 Zadání 37

3.5.2 Příprava 37

3.5.3 Využití metody 39

4 Vlastní práce **44**

4.1 Analýza 44

4.1.1 Úvod do problematiky 44

4.1.2 Otevřené otázky 44

4.1.3 Uzavřené otázky 45

4.1.4 Požadavky 46
4.1.5 Obecné požadavky 46
4.1.6 Template / od nuly 47
4.1.7 Existující plugin 48
4.1.8 Rozhodnutí postupu 49
4.1.9 Způsob napojení 49
4.1.10 Zajištění požadavků na simplexový algoritmus 50
4.1.11 Načtení parametrů a serializace 50
4.1.12 Vyhodnocení a vrácení výsledku 52
4.1.13 Návratové indexy a hodnoty 52
4.1.13.1 Hodnota účelové funkce 52
4.1.13.2 Hodnota řádku zj-cj 53
4.1.13.3 Průsečíky tabulky 53
4.1.13.4 Hodnoty b stran 54
4.1.13.5 Maximální možná hodnota b pro proměnnou 54
4.1.13.6 Hodnota Pivota 54
4.1.13.7 Jméno bazické proměnné 55
4.1.14 Návrh tříd 55
4.1.14.1 Qtype_formulas_simplex_table 55
4.1.14.2 Qtype_formulas_simplex 55
4.2 Implementace 56
4.2.1 Kontrola vstupů 56
4.2.2 Třída qtype_formulas_simplex_table 59
4.2.3 Třída qtype_formulas_simplex 59
4.2.3.1 Inicializace 60
4.2.3.2 Unit_matrix 60
4.2.3.3 Possible_index_precalculate 61
4.2.3.4 Reverse_zparameters 62
4.2.3.5 Create_table 63
4.2.3.6 Calculate 63
4.2.3.7 Decision 64
4.2.3.8 Recalculation 66

4.2.3.9 Return_answer_index 68
4.2.4 Přidání dalších indexů 70
5 Výsledky a diskuse 71
5.1 Testování 71
5.1.1 Testování funkčnosti pluginu 71
5.1.2 Testování funkčnosti nadstavby 71
5.2 Základy využití 71
5.3 Základní požadavky na uživatele 72
5.4 Příklad pro fixní hodnoty 73
5.5 Příklad pro náhodné hodnoty 77
5.6 Další vývoj 84
6 Závěr 86
7 Seznam použitých zdrojů 87
8 Seznam obrázků a kódů 91
8.1 Seznam obrázků 91
8.2 Seznam kódu 92
Přílohy 94

1 Úvod

Ověřováním znalostí a dovedností pomocí papíru a tužky si během svého studia prošel asi každý student. Už méně studentů si vyzkoušelo testování vědomostí online. S testováním znalostí simplexového algoritmu se nejčastěji setkávají studenti matematických či IT oborů vysokých škol. Dle dat českého statistického úřadu bylo v roce 2020 takovýchto studentů necelých 44 tisíc, což tvoří 14,5 procenta z celkového počtu studentů navštěvujících vysokou školu.

Pro všechny studenty je nutné vytvořit testovou úlohu v ideálním případě s odlišnými hodnotami, a přesto se stejnou výpočetní složitostí v testové skupině. Tímto procesem vytváření a opravování takovýchto úloh se tato práce zabývá.

Testování studentů v LMS Moodle je dnes běžnou praxí ať už na středních či na vysokých školách v kombinaci se zkoušením ústním, proto podpora pro testování obecných otázek je velmi dobrá. Problém nastává při specializaci otázek na konkrétní problematiku. Pro testování simplexového algoritmu v LMS Moodle neexistuje žádný přímo podporující plugin usnadňující zadávaní a vyhodnocování takovýchto úloh.

Práci na téma simplexový algoritmus a jeho testování jsem si vybral z důvodu řešení reálné problematiky pro reálné lidi. Naprostá většina prací, které jsem během svého studia vypracoval byla díla vytvořena za účelem ověření znalostí. Po výsledné kontrole kvality byla uložena nebo zahozena.

Práce je rozdělena na teoretickou a praktickou část. V teoretické části je popsán aktuální stav poznání testování v papírové a online formě. Následně je vysvětlena problematika a podpora testování přímo v LMS Moodle. V dalších částech jsou představeny jednotlivé technologie a postup výpočtu simplexového algoritmu.

V praktické části práce je provedena analýza testování simplexového algoritmu a na základě nalezených skutečností je vybráno místo pro časovou optimalizaci testování. Pro toto místo je navrženo softwarové řešení a následně je implementováno.

Výsledkem práce je softwarové řešení časové náročnosti testování simplexového algoritmu v LMS Moodle.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce se zabývá analýzou, návrhem a implementací otázkového modulu/ů pro zrychlení a zjednodušení zadávání úloh na simplexový algoritmus do LMS Moodle. Tento modul/y bude/ou řešit otázky otevřené, uzavřené a uzavřené s možnostmi.

2.2 Metodika

Diplomová práce se sestává ze dvou částí. Metodika zpracování teoretické části práce je založena na studiu odborných a vědeckých informačních zdrojů. Na základě syntézy zjištěných požadavků budou formulována teoretická východiska pro zpracování praktické části práce.

Praktická část práce spočívá v návrhu a implementaci modulu/ů. Při návrhu a implementaci bude využito standardních metod a nástrojů softwarového inženýrství. Knihovna bude prakticky otestována a budou shrnuty zásadní poznatky zjištěné při její implementaci a testování a budou navrženy možnosti případného dalšího budoucího rozšíření.

3 Teoretická východiska

3.1 Současný stav testování

V této kapitole je proveden rozbor aktuálního stavu problematiky testování znalostí v papírové a online formě a následně testování řešení simplexových algoritmů v LMS (e-learningový systém) Moodle.

3.1.1 Úvod do testování

Papírové testy vynikají svojí jednoduchostí. V naprosté většině papírových testů studentovi stačí papír a psací pomůcka. Další výhodou je bezpečnost papírových testů. Příkladem mohou být státem provozované maturitní zkoušky, kdy jsou zadání distribuována až na poslední chvíli a studenti vyplňují test pod dohledem místních učitelů. Takto je zajištěna stejná obtížnost pro všechny studenty a zároveň je zajištěna bezpečnost proti podvádění studentů.

První výhodou online testu je jeho údržba a případná změna. Na rozdíl od papírového testu je jednoduše měnitelný a stejná úloha se dá jednoduše klonovat s jinými čísly. Druhou nejrazantnější změnou je případné automatické opravování otázek v testu. Toto je pravdivé v naprosté většině testů s uzavřenými otázkami, kde počítač dokáže jednoduše porovnat studentovu zadanou hodnotu oproti správně hodnotě vyplněnou zadavatelem testu. Tato funkcionalita zobrazování vyhodnocených testů je obvykle plně nastavitelná. V případě otevřených úloh je tato problematika složitější, a i když existují nástroje strojového opravování otevřených otázek, je standardem tyto otázky nechat na opravu učiteli. Je možné zobrazovat jednotlivé statistiky jednotlivých otázek v testech, či celých testů. Další výhodou online testu je možnost nechat studenty psát test odkudkoliv a kdykoliv. Tato výhoda s sebou ovšem nese riziko integrity testu. Školy s tímto faktem bojují ať už vyžadováním zapnutí kamer na studenta (omezí tím kdykoliv složku testu) či implementací takzvaných testovacích center, kam student může kdykoliv přijít a nechat se otestovat za dozoru personálu (omezí kdekoliv). Jednou z méně známých výhod online testování je možnost nechat studenty si zkusit cvičné testy. Na cvičných testech si studenti nejen daný testovací systém vyzkouší a přijdou na jeho ovládání, ale také si ověří své znalosti před testem a mohou se cíleně zaměřit na nedostatky ve své přípravě. Poslední výhodou online

testování je připravenost pro studenty se speciálními potřebami. Například místo psaní odpovědí je možné použít speech to text (převodník mluveného slova do textu) či si zvětšovat písmena v testu a spoustu dalších možností.

Pojďme se nyní podívat na nevýhody online testů. První nevýhodou je, že technologie může selhat a mohou nastat různě závažné chyby. Další nevýhodou je obvykle špatná technická vybavenost učitelského sboru a s tím související nutná pomoc s vytvářením a správou testů.

Kdybychom se podívali jen na parametry obou druhů testování vychází na první pohled online testování ze všech parametrů kromě bezpečnosti lépe. Bylo by tedy logické, že studenti dosahují lepších výsledků z pohodlí svého domova a v čase, který si pro test vyberou. Ze studií na téma online testování vychází ovšem zajímavé poznatky. Prvním poznatkem je, že kdybychom vzali 2 studenty stejných znalostí a jeden vyplňoval test online a druhý test v papírové formě, tak by například v matematice student vyplňující test online byl o 0,1 standardní odchylky horší než student vyplňující test na papíru. Ačkoliv se online test napodobuje test papírový některé ovládací prvky online testu se jednoduše odlišují. Příkladem mohou být dlouhé texty na pochopení významu. Zatímco v online verzi student musí posouvat textem, tak v případě papírového testu má student stránky fyzicky v ruce a otáčí je. Studie nicméně v experimentu pokračovala i další roky a zjistila, že postupem času se rozdíly mezi oběma typy testů zmenšovali. K tomuto faktu, že typy testů jsou statisticky zaměnitelné došla i skupina německých výzkumných pracovníků. Mimo tento poznatek se zaměřili na chování jednotlivých studentů při vyplňování testů. Zjistili, že studenti vyplňující test online potřebovali k testu znatelně méně času než studenti vyplňující test papírový. Tento efekt byl především znát na studentech s lepšími výsledky. Dalším poznatkem bylo, že studenti s horšími výsledky v online verzi testu hádali odpovědi mnohem častěji než studenti vyplňující papírovou verzi testu. [1-8]

3.1.2 Testování simplexového algoritmu

U testování simplexového algoritmu je potřeba otestovat studentovu schopnost převést data z textu na informace a na jejich základech vytvořit podmínky a účelovou funkci. V dalším kroku musí student prokázat znalost získání základního řešení a následné využití simplexového algoritmu pro nalezení optimálního řešení. Poté student musí své výsledky interpretovat. V rámci testování jsou běžné teoretické otázky souvise-

jící se simplexovým algoritmem stejně jako postoptimalizační otázky. Problémem při testování všech těchto částí bývá časová náročnost testu, která je způsobená především manuálním počítáním simplexového algoritmu. Je běžné, že student je nucen provést několik iterací a každá iterace v závislosti na množství rovnic a proměnných trvá v jednotkách desítek minut na provedení. Některé instituce proto testování simplexového algoritmu mají jako samostatný test, na který studentům dají čas v hodinách a poté mohou otestovat důkladně všechny aspekty studentových znalostí. Druhým přístupem k tomuto problému jsou otázky na konkrétní rozhodnutí v rozhodovacím kroku simplexového algoritmu. Student nemusí počítat celé kroky, ale pouze část kroku. Příkladem takové otázky může být výběr, která proměnná do báze vstoupí a která proměnná z báze vystoupí nebo obecná otázka, jaký je další krok postupu. Toto řešení problému časové náročnosti také standardizuje odpovědi studentů a ty potom mohou být vyhodnoceny automaticky. U testování celých příkladů se studenti mohou vydat neoptimální cestou řešení a stejně mohou dojít k správnému výsledku. [9,10]

3.1.3 Testování simplexového algoritmu v LMS Moodle

Ačkoliv LMS Moodle, jako jednička na trhu vyvinula desítky addonů a další tisíce addonů jsou pro Moodle vytvořeny komunitou, tak nemá rozšíření přímo podporující testování úloh simplexového algoritmu. Test v LMS Moodle se skládá ze stejných částí jako test písemný. Výroba simplexových úloh za současné podpory LMS Moodle vede k následujícím krokům. Nejdříve zadavatel vytvoří otevřené či uzavřené otázky testující teoretické znalosti.

Následně vytvoří specifický test na jedno použití pomocí hrubé síly. V tomto případě profesor postupoval vytvořením první otevřené otázky, ve které simplexovou úlohu zadal se stejnými čísly pro všechny studenty. Tuto úlohu bude sám po vyplnění studentem opravovat a je v ní příprava, výpočet plus interpretace výsledků simplexového algoritmu. Následně si k této otázce vytvoří otevřené postoptimalizační podotázky, ze kterých Moodle náhodně generuje do testu nebo si profesor vyrobí jen otázky, které potřebuje do testu a studenti mají všichni stejný test. Tyto otázky vyhodnotí LMS Moodle sám. V tomto postupu profesor může vyrobit více takových testů a LMS Moodle mu umožní náhodně vybrat, jaký test bude studentovi přiřazen. Nicméně veškeré výpočty úloh při přípravě musí být provedeny profesorem mimo Moodle a obvykle v nějaké simplexové kalkulačce

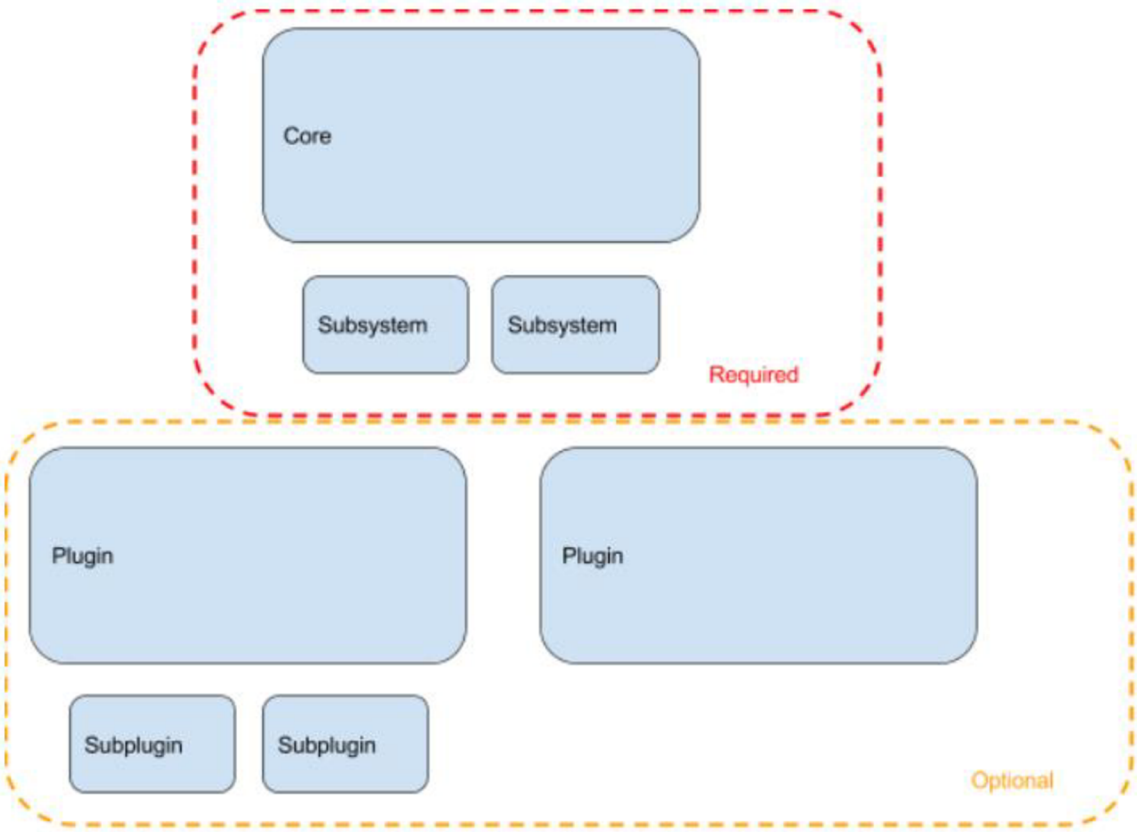
z internetu nebo v excelu či na papíře. Jednou z úprav tohoto postupu je vytvoření několika takovýchto otázek a využití funkce Moodle k vybrání jedné náhodné verze pro studenta při zahájení testu.

3.2 LMS Moodle

Moodle je open source systém LMS napsaný v PHP. Moodle je acronym modular object-oriented dynamic learning environment v překladu modulární objektově orientované dynamické učební prostředí. Původně písmeno M bylo v názvu podle zakladatele Martina Dougiamase. Moodle vznikl v roce 1999 a v průběhu si prošel spousty změn včetně v roce 2001 změnou architektury, která zůstala dodnes. Dnes Moodle podporuje stovky různých jazyků a využívají ho univerzity, státní sektor i soukromý sektor. [11]

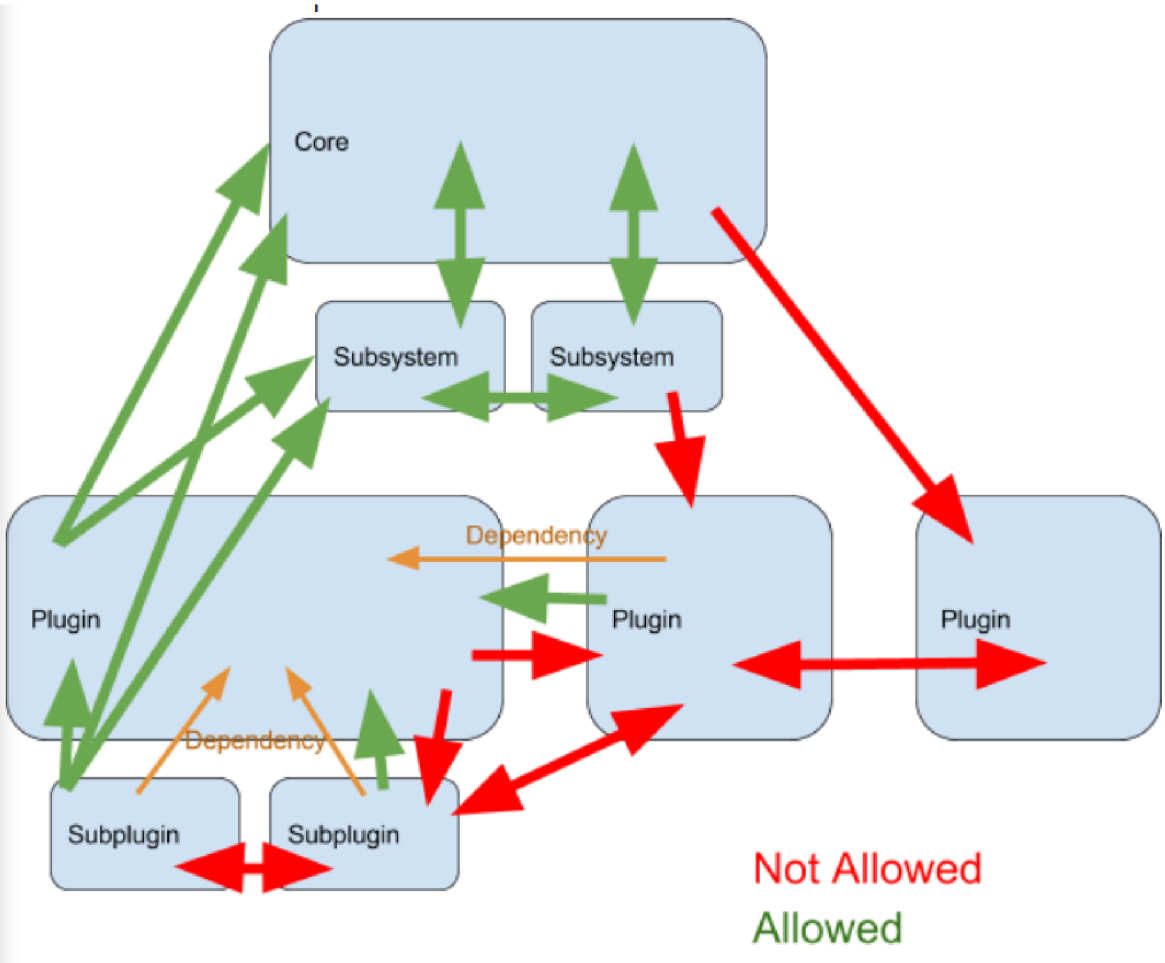
3.2.1 Moodle struktura

Moodle se skládá z jádra (Core), což je část kódu, která zaručuje základní funkcionalitu Moodlu. Součástí jádra jsou také jednotlivé subsystémy (Subsystem), což jsou třídy a funkce k obsluze jednotlivých funkcionalit Moodlu. Ačkoliv tyto funkce mohou občas být v konfiguračním souboru deaktivovány (či následně znovu aktivovány) obecně jsou volně zavolatelné z jakéhokoliv modulu Moodlu. Na jádro jsou dále navázány pluginy, které dále rozvíjejí funkcionalitu Moodle od bezpečnosti, vizuální podoby, přes hodnocení testů, vytváření otázek a mnoho dalších typů pluginů. Některé pluginy mohou mít také svoje sub-pluginy, které dále rozvíjejí daný plugin a jsou tak na něm závislí.



Obrázek 1: Komunikace mezi komponenty [12]

Protože všechny pluginy jsou volitelné, a nemusí být nainstalovány, tak Moodle zavedl několik pravidel pro komunikaci mezi komponenty. Je povoleno komunikovat přímo s jakýmkoliv modulem směrem vzhůru stejně tak, jako komunikovat sám se sebou v rámci modulu. Dále je povoleno komunikovat s jakýmkoliv modulem, pokud modul, ze které volám je závislý na volaném modulu. Všechny jiné volání jsou zakázané. [12]



Obrázek 2: Povolená komunikace v Moodle [12]

3.2.2 Otázka

Otázka v LMS Moodle může nabývat různých podob v závislosti na svém typu. Tento typ otázky určuje její vlastnosti a možnosti volitelných parametrů. Pro vysvětlení základních funkcionalit a principů společných pro všechny typy otázek si rozebereme nejjednodušší typ otázky známou v Moodle pod názvem qtype_essay.

Qtype_essay je složen z textu otázky a po studentovi v testu je požadována otevřená odpověď či soubor. Pro vytvoření otázky qtype_essay musí zadavatel vybrat kategorii, zadat název a text otázky.

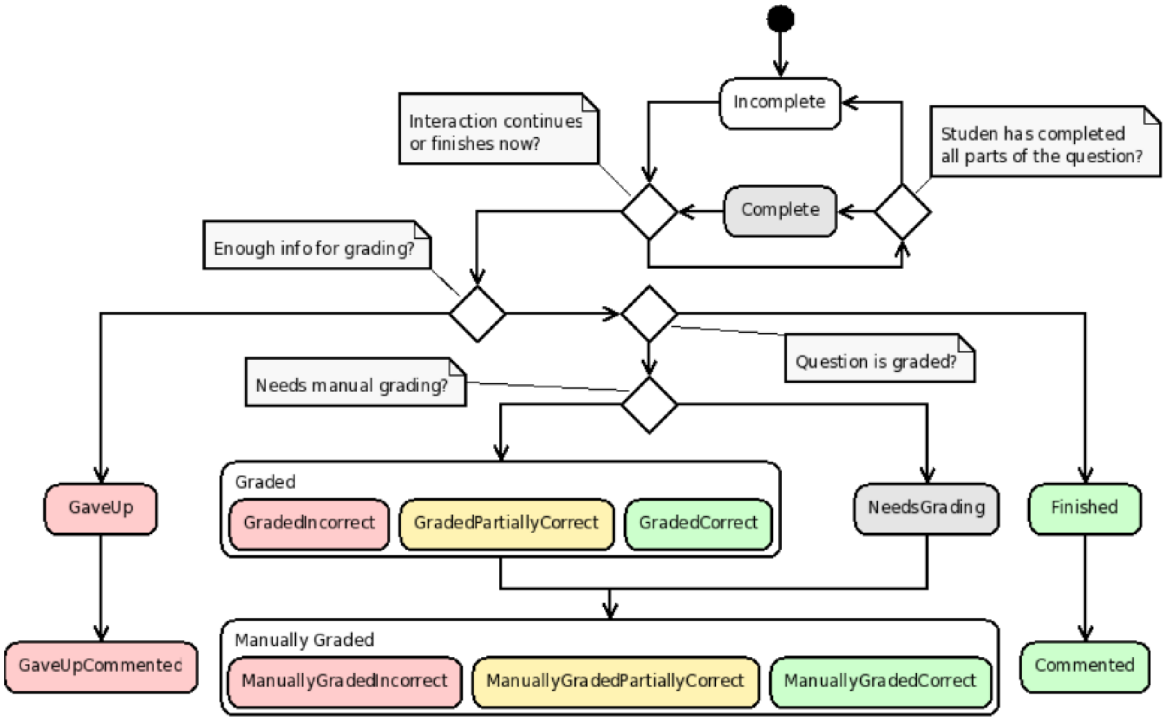
Další parametry otázky jsou nepovinné. Může nastavit konkrétní identifikační číslo otázky a musí nastavit povinnost či nepovinnost odpovědi její délky a zobrazení. Tato sekce je předvyplněná defaultními hodnotami. Je možné také předvyplnit text, který se v poli

pro odpověď studentovi zobrazí. Tímto způsobem je možné studentovi poskytnout osnovu požadované odpovědi. Dále je možnost vyplnit pole pro hodnotitele testu, kdy je možné hodnotiteli testu zobrazit například modelovou odpověď na otázku či výsledek úlohy. Stejně tak je možné označit otázku tagem. Rozdíl mezi kategorií a tagem je jeho využití. Kategorie a podkategorie využijeme například v případě kategorie úloh matematiky a podkategorie bude rovnice, kde budeme mít úlohy týkajících se rovnic. V případě tagů nejde o fyzické rozdělení otázek do skupin, ale pouze parametr, který můžeme například využít v bance úloh k zjednodušení vyhledávání.

Otázka byla uložena a přiřazena do testu a spuštěna studentem. Aby Moodle mohl otázku zobrazit potřebuje při zobecnění následující 4 druhy informací. Druh otázky, chování otázky, v jakém stavu se otázka nachází a nastavení aktivity (v našem modelovém případě testu). Druhem otázky je například qtype_essay či jakýkoliv jiný typ otázky. Chování otázky je popsáno způsobem, jakým je otázka hodnocena. První možností je klasické vyhodnocení otázek (deferred feedback) po vyplnění a odevzdání všech otázek (vyhodnocena systémem dle předem připravené odpovědi), druhou možností je takzvaný adaptivní mód (adaptive mode), kdy je studentovi zobrazeno odevzdání každé otázky zvlášť a když ji student odevzdá a nemá správnou odpověď (vyhodnocena systémem dle předem připravené odpovědi) je mu umožněno svou odpověď změnit, ale za zhoršené bodové ohodnocení. Třetí možností je manuální hodnocení učitelem. Tato možnost se uplatní především u otevřených typů otázek. I když existují typy otázek, které se snaží opravy otevřených otázek automatizovat pomocí hledání klíčových slov zadaných učitelem ve studentově textu. Takovýchto druhů chování existují desítky. Třetím druhem informace je stav, ve kterém se otázka nachází. Následující zobecněné schéma tyto stavy popisuje. Způsob, jakým se otázka pohybuje mezi stavy (a jaké jsou) v Moodlu je ovlivněn nejen druhem otázky, ale i nastaveným chováním.

Incomplete stav je počáteční stav otázky. V tomto stavu se otázka nachází, dokud student by měl věnovat této otázce pozornost. Při deferred módu, než student odpověděl na otázku v závislosti na typu otázky. Ať už zadal text nebo vybral z možností či přetáhl správnou odpověď. V adaptivním módu otázka zůstává v tomto stavu, dokud student neodpověděl správně nebo nevyčerpal všechny své pokusy. Obecně v tomto stavu otázky může student volně měnit svoji odpověď. Dalším stavem je stav complete. Tento stav navazuje na stav incomplete a vyjadřuje splnění požadavků na odpověď. Pokud jsme v módu

deferred feedback otázka v tomto stavu čeká na odevzdání testu, ať už manuálním studentovým zásahem nebo z důvodu vyčerpání povoleného času na test či jiného důvodu. Před odevzdáním z jakéhokoliv důvodu může stále student svoji odpověď měnit. Po odevzdání ve všech další možných stavech už student nemůže svoji odpověď měnit. Prvním takovým stavem je graded. Ten shlukuje podstavy automatické opravy, pokud je možná. Výsledkem takové opravy je jeden ze stavů correct, partiallyCorrect a incorrect. V překladu správná odpověď, částečně správná odpověď a nesprávná odpověď. Druhou možností jsou otázky nutné vyhodnotit fyzickou osobou, k tomuto je stav manualGraded a podstavy téměř identické s automatickou opravou. Mohou nicméně nastat speciální případy. Prvním případem je případ popisků, kdy není možné či žádoucí přiřazovat body. V takovém případě otázka po odevzdání přechází otázka do stavu finished případně commented pokud se opravující rozhodne napsat komentář. Druhou možností je například nevyplnění otázky vůbec nebo částečné vyplnění. Tento stav se jmenuje giveUp a primárně sdružuje nevyplněné otázky studentem s možností stavu giveUpCommented, kdy opravovatel nechá studentovi komentář, například s řešením či radou pro studenta, aby se mohl ze svých chyb poučit. V případě neúplně vypracovaných otázek záleží na kontextu a typu otázky, protože v nějakém případě je záhodno neúplně zodpovězenou otázku bodově ohodnotit. Taková možnost je v stavovém grafu zanedbána.



Obrázek 3: Stavy a jejich přechody [12]

```php
abstract class question_state {
    const NOT_STARTED = -1;
    const UNPROCESSED = 0;
    const INCOMPLETE = 1;
    const COMPLETE = 2;
    const NEEDS_GRADING = 16;
    const FINISHED = 17;
    const GAVE_UP = 18;
    const GRADED_INCORRECT = 24;
    const GRADED_PARTCORRECT = 25;
    const GRADED_CORRECT = 26;
    const FINISHED_COMMENTED = 49;
    const GAVE_UP_COMMENTED = 50;
    const MANUALLY_GRADED_INCORRECT = 56;
    const MANUALLY_GRADED_PARTCORRECT = 57;
    const MANUALLY_GRADED_CORRECT = 58;

    public static function is_active($state) { ... }
    public static function is_finished($state) { ... }
    // ...
}
```

Obrázek 4: Hodnoty jednotlivých stavů [12]

Čtvrtá a poslední informace se týká nastavení dané aktivity. Tyto parametry určují například práva na zobrazení opraveného testu studentem, nebo možnost vypnutí známek a ponechat otázky vyhodnocené pouze jako správně či špatně. Tato funkčnost se nejčastěji využívá u selftestů, kde si student sám může ověřit své znalosti. Práce Moodle se známkami z testu je obecně zajímavá, neboť využívá tří pojmů volně přeložených, jako známka, skóre a body. Mějme test sestávající se z 6 otázek po 3 bodech a jedné otázce po 2 bodech. Při zisku 20 bodů student získá skóre násobením počtu bodů z otázek a v tomto případě vynásobením 5. Takto získané skóre se porovná s rozsahy známek a výsledek je zapsán studentovi do známek. Na nejnižší úrovni Moodle pracuje s body za otázku v rámci zlomků (číslo 0-1). Tento zlomek při potřebě spočítat body za otázku vynásobí maximální ziskem bodů za otázku. Celý tento proces dělení bodů za otázku umožňuje lepší změny bodů za danou otázku v konkrétním testu. [13,14]

3.2.3 Question bank

Question bank je nástroj Moodle starající o zobrazení a správu vytvořených otázek z databáze. Otázky jde vytvářet, upravovat, zobrazovat a ukládat. Otázky je možné klonovat, importovat a exportovat do a z Moodle. Moodle také implementuje systém kategorií, který umožňuje otázky třídit a následně v testech vybírat náhodně z vybrané kategorie. Tyto kategorie také mohou mít své podkategorie. Tento nástroj je přístupný pouze uživatelům s právy pro přístup. Obvykle to bývá správce a učitelé spravující si své kategorie otázek pro studenty. [15,16]

3.2.4 Question engine

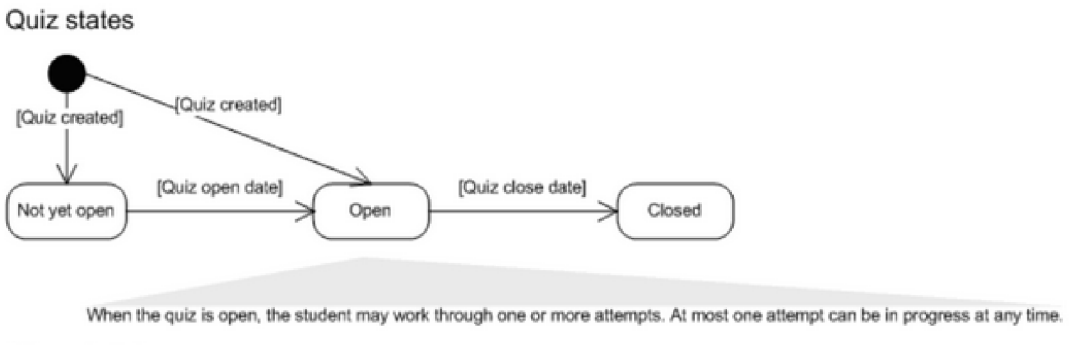
Question engine je část kódu starající se zpracování otázky od momentu, kdy ji učitel přiřadí do testu a student test spustí. Celý test si můžeme představit jako soubor jednotlivých otázek. Pro každý pokus musíme vědět v jakém stavu se daná otázka nachází a informace o hodnocení či typových charakteristikách. Příkladem je například míchání pořadí odpovědí u uzavřených otázek s možnostmi či generování náhodných čísel pro otázky a další. Question engine také zajišťuje zobrazení otázek nejen v průběhu testu, ale

také po dokončení testu, kdy dle nastavení umožní studentovi ihned nahlédnout do svého testu či až po opravě učitelem nebo mu to neumožní vůbec.

Pro případ testu (quiz modul) průběh vypadá následovně. Pro vyrenderování otázek v testu by byly zavolány metody load_questions, load_question_states a podle množství otázek v testu print_questions. Součástí těchto funkcí je delegace části zobrazení či přípravy do kódu danému typu otázky (například qtype_essay). V průběhu testu student vyplňuje jednotlivé otázky a tím mění stavy a hodnoty jednotlivých odpovědí. Moodle každou změnu otázky zaznamenává pomocí objektu zvaného question_attempt ten se skládá z jednotlivých question_attempt_steps. Nadřazený objekt pro celý pokus testu se jmenuje question_usage_by_activity. Následuje vyhodnocení. Zde se zavolají metody dle nutnosti manuální opravy učitelem. V případě strojové opravy jsou to funkce load_questions, load_question_states, question_extract_responses, question_process_responses, save_question_state a v případě testu se následně uloží sloupec v databázi se jménem quiz_attempts. Tento sloupec byl upraven volanými metodami a stejně jako při inicializaci některé úkony byly delegovány do třídy typu otázky. Instance typu otázky je singleton (má pouze jednu instanci). Ta se stará o zpracování všech otázek svého typu v testu. V případě manuální opravy učitelem jsou nahrazeny funkce question_extract_responses, question_process_responses funkcí question_process_comment. [14,17,18]

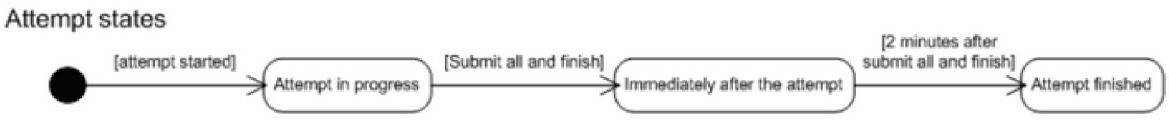
3.2.5 Quiz

Test se může nacházet po vytvoření v několika stavech. Pokud datum otevření testu ještě nenastalo. Není možné test studentem spustit a je mu zobrazen datum a čas otevření testu. V moment uplynutí nastaveného datumu testu je studentovi test zpřístupněn. Je možné nastavit i čas uzavření testu. V takovém případě se student může pokusit o zvládnutí testu pouze mezi těmito datumy.



Obrázek 5: Quiz stavy [23]

Je možné nastavit možný počet pokusů (attempt) studenta na test, stejně jako časový rozestup mezi jednotlivými pokusy. Vždy může být aktivní pouze jeden pokus. Tyto a další parametry jsou zvoleny při vytváření testů případně po vytvoření následnou editací. Pokus začíná od studentova spuštění testu. Student vyplní test a buď odevzdá test nebo je za něj odevzdá systém při vypršení času. Nyní pokus přejde do mezistavu než je uložen. V něm čeká, než je uložen. Tento mezistav řeší problém vytížení zdrojů serveru při situaci, že velký počet studentů odevzdá svůj pokus najednou. V teoretické rovině by se mohlo stát, že by se systém snažil uložit pokus po vyčerpání času na test. V některých verzích Moodle je tento čas fixně nastaven na 2 minuty jinde flexibilně na 5 procent z celkového času na test. [19-24]



Obrázek 6: Pokus stavy [23]

3.2.6 Moodle question type

Otázkové moduly musí být ve složce question/type/. Rozvržení uvnitř této složky odpovídá typickému rozvržení pluginu Moodle. Například ve složce question/type/název_typu/ bychom měli:

- question.php
- edit_*Název*_form.php

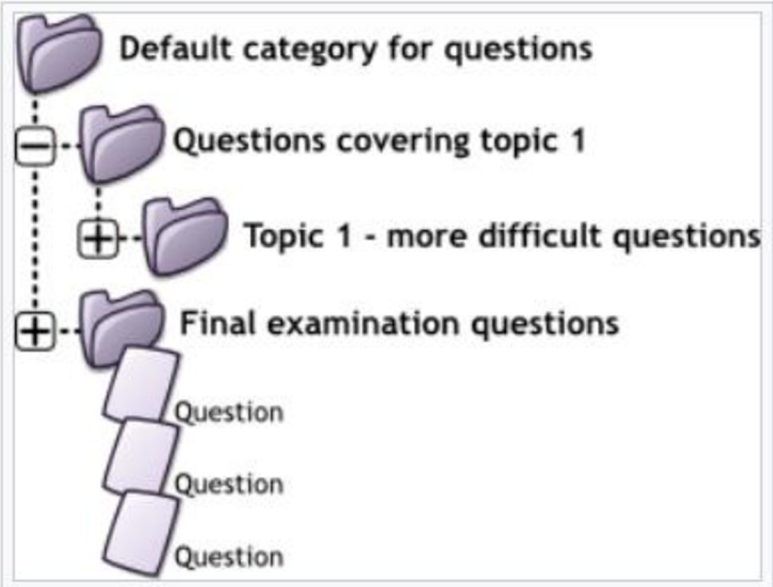
- question_type.php
- renderer.php
- složka pro testy
- lang složka pro jazykové variace
- version.php
- složka pro definici databázové struktury otázky
- settings.php
- a další

Soubor question.php obsahuje definici dané otázky. Pro každou otázku je vytvořena jedna instance této třídy. Tato třída není jen definicí otázky, může také sledovat aktuální stav otázky, když se o ni student pokouší. Třída question_type naproti tomu zajišťuje metadata, stejně jako obsahuje metody, jak pracovat s instancemi otázky. Určuje, jak se otázka bude načítat, ukládat, mazat a obecně pracovat s instancemi otázky. Plugin obvykle obsahuje složky pro definování struktury databáze pro hodnoty, které nepokrývá question_definition nebo jeden z jejích potomků. Ve stejné složce můžeme najít odinstalační metody či metody pro upgrade. Dále obsahuje soubory, ve kterých jsou popsány metody pro zobrazování HTML výstupů (renderer.php), stejně jako složku pro testy. V settings.php můžou být určeny defaultních hodnoty či jiné parametry nastavení funkčnosti. Jedním z posledních důležitějších souborů je version.php obsahující informace o aktuální verzi otázkového modulu a informace o požadovaných jiných modulech, které otázkový modul potřebuje ke správnému fungování. V otázkovém modulu existují i další soubory jejichž kód obstarává další funkcionality, jako styly, přídavné třídy funkčnosti, či složka pro JavaScript moduly. [25]

3.2.7 Kategorie testových úloh

Kategorie testových úloh, do kterých je celé úložiště rozděleno, funguje podobně jako adresáře souborů v počítači. Účelem je zpřehlednit obsah banky úloh a umožnit rychlé hledání konkrétní úlohy. Jednotlivé kategorie krom otázek mohou obsahovat také podkategorie. Každá kategorie nebo podkategorie musí mít svůj název a může obsahovat popisek a rodičovskou kategorii. Tento způsob je doporučován, protože při sestavování testu je

možné zvolit náhodnou otázku z kategorie nebo podkategorie. V případě jedné složky obsahující nehomogenní otázky tato funkce pozbývá smyslu. Nebo ho alespoň limituje.

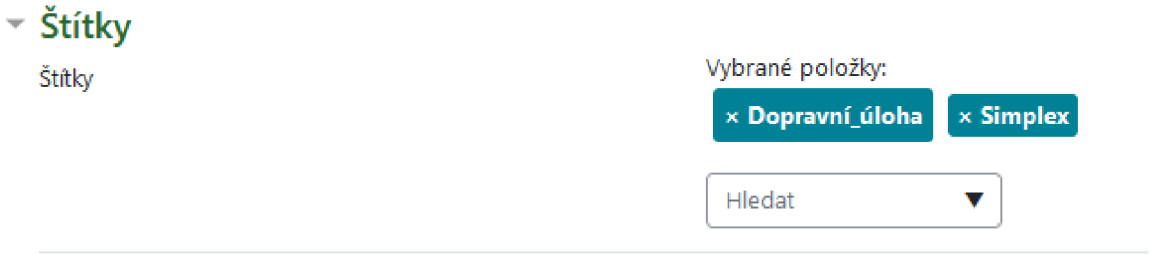


Obrázek 7: Kategorie struktura [26]

Stejně jako u práv na složky u Windows i zde se uživateli systému přiřadí defaultní nebo nastavená práva pro viditelnost složek. Obvyklým stavem je, že profesor vidí jen své kategorie otázek spadající pod ním vyučované předměty [26-27]

3.2.8 Štítky

Podpora štítků pro otázky byla přidána do Moodle ve verzi 3.6. Štítek vlastní informace o svém názvu, popisu a tvůrci, stejně jako potenciální vazbu na jiný štítek. Tato funkcionalita zajišťuje možnost zaznamenávat k jakému tématu se daná otázka vztahuje a tím je sdružovat do skupin. Následně je možné štítky vytvářet či propojovat a v otázce vybrat jeden či více štítků k dané úloze.



Obrázek 8: Štítky v úloze [vlastní zpracování]

Pomocí štítků lze také filtrovat úlohy v bance úloh a nejdříve si úlohy daného typu vyhledat a následně je možné do testu buď vybrat konkrétní úlohu nebo náhodnou úlohu vlastnící daný štítek nebo kombinaci štítků. [28]

Obrázek 9: Štítky v bance úloh [vlastní zpracování]

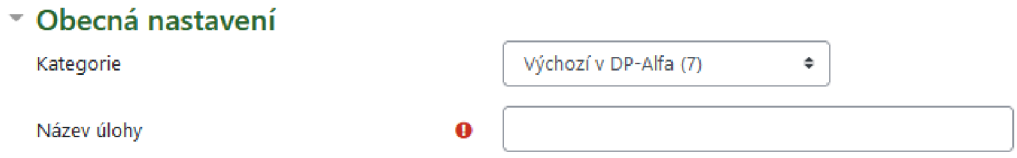
3.2.9 Formulas

Plugin Formulas je obecný otázkový plugin zaměřující se na zjednodušení testování netriviálních testových otázek z oblasti STEM (věda, technika, inženýrství a matematika).

Jeho struktura umožňuje vysokou flexibilitu definování otázek a obecné zjednodušení vytváření. Formulas umožňuje následující funkce:

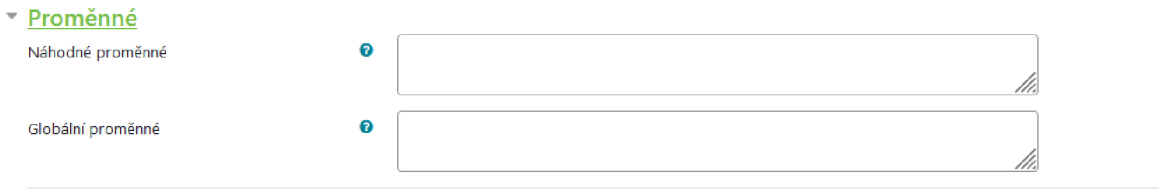
- **Náhodné proměnné** Každý student může dostávat otázky s jinými čísly.
- **Více částí otázky** Plugin umožňuje vytvářet více otázek se stejnými náhodnými čísly.
- **Více jak 1 odpověď** Možnost více částečné odpovědi.
- **Různé typy podporovaných odpovědí** Lze použít jak numerické odpovědi. s jednotkami, tak algebraické odpovědi.
- **Možnost chyby výsledku** Je možné zadat rozsah správné odpovědi.
- **Jednotky** Je možné zadat jednotky odpovědi, které musí student zadat.
- **Možnost pokusů** Pro každou část otázky v rámci probíhajícího testu lze zadat počet povolených pokusů.

Zadávání obecné Formulas otázky vypadá následovně: První část tvoří obecná nastavení složená z kategorie otázky a názvu úlohy.



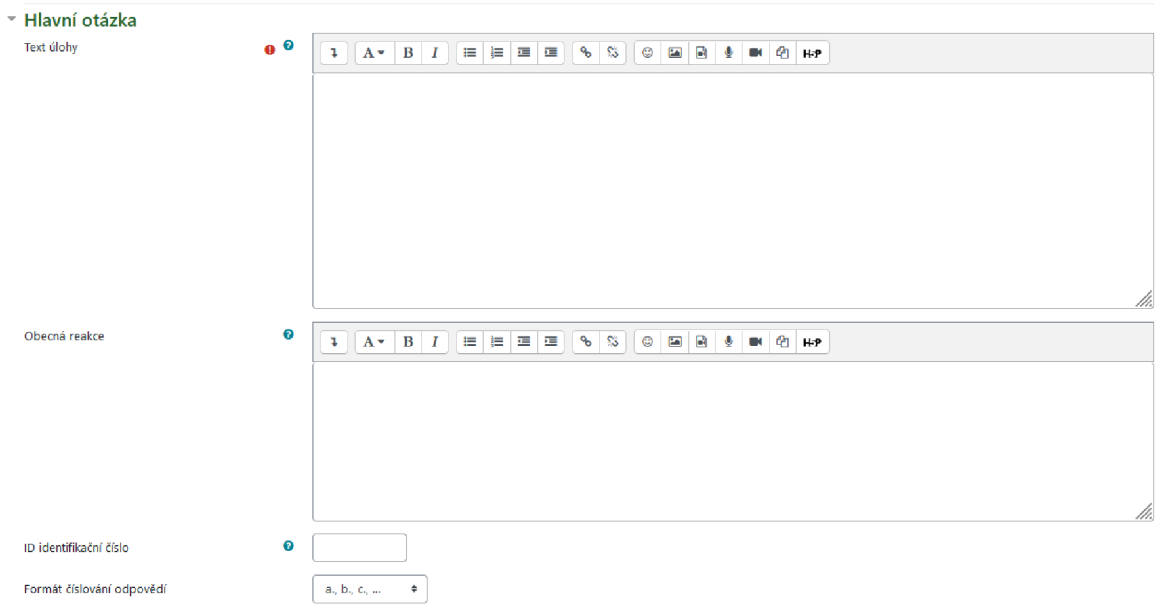
Obrázek 10: Formulas obecná nastavení [vlastní zpracování]

Následná část je zaměřena na zadání náhodně generovaných proměnných a následně globálních proměnných. Hodnoty takto definovaných proměnných nabývají pro celou otázku stejné hodnoty. V rámci formulas nicméně je možné definovat i lokální proměnné, které jsou specifické pro každou část otázky. Existují ještě pomocné hodnotící proměnné. Proměnné jsou nazývány řetězci alfanumerických znaků (a-z, A-Z, 0-9 a _) a nesmí začínat číslem či podtržítkem. Proměnné také rozlišují velká a malá písmena. Každá proměnná také musí nabývat jednoho z pěti typů. Čísla, řetězce znaků, listu čísel, listu řetězců znaků nebo algebraické proměnné. Každý z těchto typů má svůj způsob zápisu, stejně tak jako se odlišuje zápis náhodných proměnných a proměnných globálních. Formulas také podporuje úpravu hodnot funkcemi a řídícími prvky (například cyklus for) v poli globálních proměnných.



Obrázek 11: Formulas proměnné [vlastní zpracování]

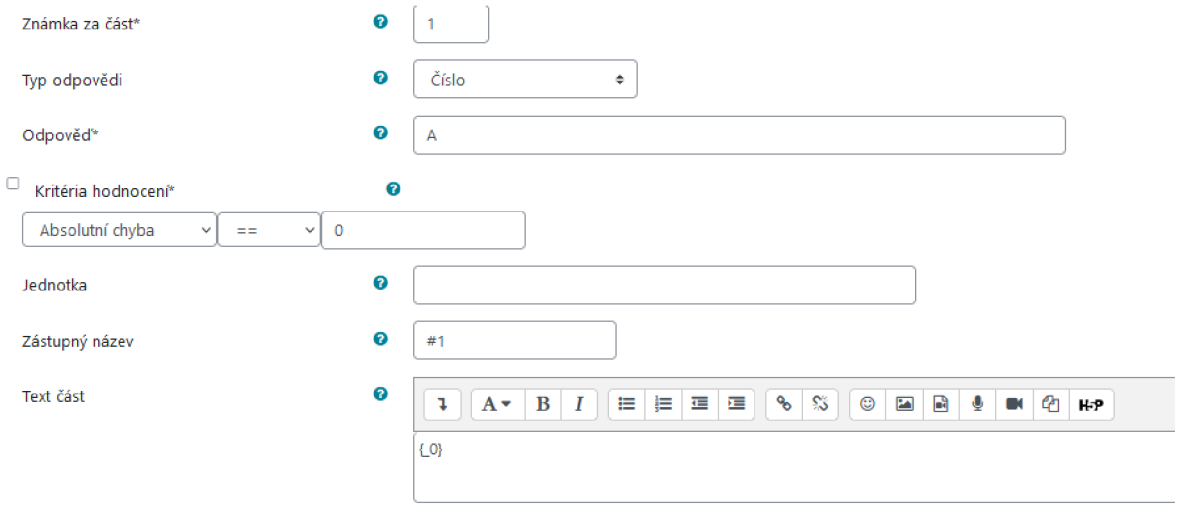
Další částí Formulas otázky je úvodní text úlohy V této části je obecné zadání úlohy s odkazy na proměnné z předchozí části a případně zástupné symboly pro části otázky (o tomto více v další části). Dále obsahuje obecnou reakci a identifikační číslo. Poslední částí je zvolení způsobu číslování jednotlivých částí otázek.



Obrázek 12: Formulas úvodní text [vlastní zpracování]

V další části jsou popsané jednotlivé otázky. Každá taková podotázka musí mít svoji známku rozdílnou nule (výsledná známka je součet podotázek), typ odpovědi, který po studentovi bude požadován a samotná odpověď, oproti které systém vyhodnocuje správnost studentovi odpovědi. Dále je možné nastavit u číselných odpovědí rozmezí správné odpovědi a jednotku, kterou musí student vyplnit. Dále Formulas umožňuje označit jednotlivé podotázky zástupným názvem a tuto značku umístit do úvodního textu. Tato značka je

při zobrazení studentovi zobrazena jako text z pole Text část. V případě nevyplnění značky jsou jednotlivé části zobrazeny pod sebou v pořadí, jaké mají číslo podotázky. Formulas dále podporuje systém průběžného vyhodnocení, kdy je možné nastavit jednotlivé rady pro studenty v případě špatné odpovědi a tím je nasměrovat ke správné odpovědi.



Obrázek 13: Formulas podotázka [vlastní zpracování]

Formulas dále obsahuje několik dalších částí. Jmenovitě část pro převodní pravidla, kontrolu instance proměnných, kde je možné si celou otázku zobrazit stejně jako hodnoty a statistiky proměnných a jejich celé instancované sady. Dále obsahuje část pro správu štítků otázky a část zobrazující datum a osobu, která konkrétní otázku vytvořila a upravila. [29]

3.3 PHP

PHP (PHP: Hypertext Preprocessor) je skriptovací programovací jazyk. Je určený pro dynamické i statické webové stránky nicméně je možné v PHP napsat konzolové či desktopové aplikace. PHP umí přijímat data z formulářů, generovat dynamický obsah stránek, pracovat s databázemi, vytvářet relace, odesílat a přijímat soubory cookies, odesílat e-maily atd. V PHP je také k dispozici mnoho hashovacích funkcí pro šifrování uživatelských dat, díky nimž je PHP bezpečné a spolehlivé pro použití jako skriptovací jazyk na straně serveru. [30]

3.4 Operační analýza

Operační analýza se zabývá uplatněním matematický metod k řešení zejména ekonomických, logistických či vojenských úloh. Cílem je vytvořit model daného problému a následně být schopný takovou situaci optimalizovat, tedy najít takové parametry, aby výstup modelu byl požadovaný extrém čili optimum. Tento teoretický základ pochází z obecné optimalizační úlohy (Někdy známé pod úlohami matematického programování). Optimalizačních úloh a jejich typů je spousta nicméně pro tuto práci nám bude stačit úsek lineárního programování.

První náznaky myšlenek souvisejících s lineárním programováním pochází od Jeana Baptisty Josepha Fouriera, který žil na přelomu 19 století. Jeho studium řešení soustav nerovnic z teoretické mechaniky nicméně příliš daleko nevedlo. Zatímco Jean Baptista Joseph Fouriera se zabýval nerovnicemi Carl Friedrich Gauss studoval řešení lineárních rovnic. Za zmínku stojí především eliminační algoritmus pod jménem Gaussova eliminační metoda nebo také Gaussova eliminace. Ačkoliv tu byli i další vědci se svými díly přesuneme se k dílu Theodora Samuela Motzkina, který ve své disertační práci Beiträge zur Theorieder linearen Ungleichungen (volně přeloženo jako příspěvky k teorii lineárních nerovností). V této práci je shrnuto do té doby 42 publikovaných děl zaměřených na problematiku lineárních nerovnic. Jeho dílo sleduje jak analytický, tak geometrický ráz problematiky.

První zmínky o lineární optimalizaci v dnešním významu pochází od ruského matematika Leonida Vitaleviče Kantoroviče z roku 1939. Bohužel toto předválečné období pro jeho práci nebylo nejšťastnějším a využití jeho práce se věnovali státy především po ní. Před publikací obecné metody pro řešení úloh lineárního programování byli řešeny dílčí problémy, jako kombinatorické řešení přiřazovacího problému či řešení dopravního problému. Za zmínku také stojí kniha Theory of Games and Economic Behaviour (Teorie her a ekonomického chování) napsanou Johnem von Neumannem a Oskarem Morgensternem

V roce 1947 George Bernard Dantzig pro potřeby amerického letectva vyvinul metodu schopnou obecně řešit úlohy lineárního programování. Dostala název simplexová metoda. Jedná se o využití Jordanovi modifikace Gaussovy eliminační metody vylepšené o účelovou funkci. Simplexová metoda byla tehdy testována na tzv. dietní problém. Jde o problém, kdy se snažíme do jídelníčku zařadit předepsané množství živin, a přitom udržet cenu potravin na minimu. Šlo o soustavu 9 rovnic a 77 proměnných. takovýto problém tehdy trvalo Dantzigově týmu 120 dní práce jednoho člověka. Z dnešního pohledu by takovýto úkol pro výpočetní sílu dnešních počítačů byl směšný. [31-33]

3.4.1 Lineární programování

Před podrobnějším popisem lineárního programování bychom si měli ujasnit přednosti i nedostatky operačního výzkumu, do kterého lineární programování spadá. Operační výzkum se dnes tříští na spoustu samostatných disciplín nicméně společným rysem jsou modelové techniky. Ve zkratce máme reálný systém a my podle něho vytvoříme model. Je nutné si uvědomit podstatnou vadu, která vyplývá z tohoto kroku. Ať by se výrobce modelu snažil sebevíc, vždy bude model pouze abstraktním zjednodušením reality. Tento krok sebou ovšem nese také pozitivní vlastnosti. Můžeme popsat reálný systém v jakémkoliv stavu například zamýšleném. Dále model urychluje chování systému, takže je možné vyhodnocovat změny v systému za dlouhou dobu, stejně jako je možné s modelem provádět opakované pokusy s jinými parametry bez finančních ztrát na rozdíl od reálného systému. [34,35]

3.4.2 Postup řešení úlohy lineárního programování

Postup řešení lze shrnout do několika bodů:

- Formulace problému (přesné slovní vyjádření)
- Formulace matematického modelu
- Řešení matematického modelu
- Interpretace řešení
- Postoptimalizační analýza

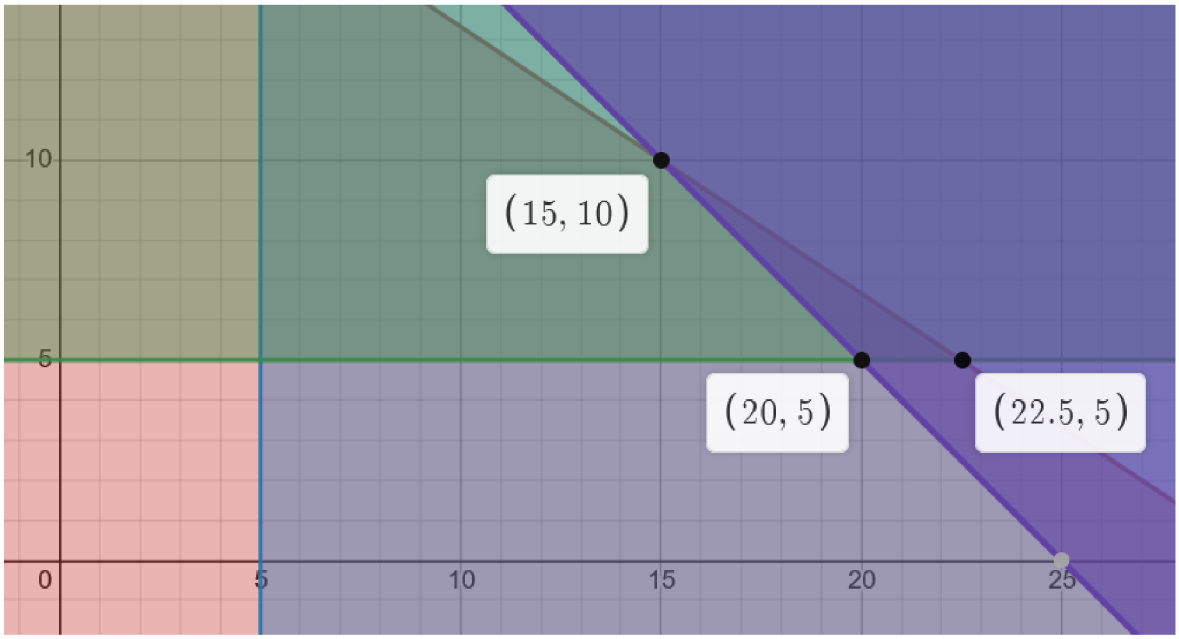
Tento postup provedeme na příkladu firmy vyrábějící 2 druhy papírových beden. Firma vyrábí bedny A za 2 minuty a prodává je s čistým ziskem 4 Kč. Bednu B vyrobí za 3 minuty nicméně má z ní čistý zisk 5 Kč. Klient si objednal minimálně 25 beden s podmínkami, že chce alespoň 5 beden od každého druhu a zároveň je chce mít vyrobeny do hodiny. (Pro náš příklad zanedbáme sklad s předem vyrobenými bednami. Bereme v úvahu, že objednávku musíme vyrobit za udanou 1 hodinu 1 pracovník. Jaká bude kombinace beden A a B, aby firma měla co největší čistý zisk?

Tímto popisem jsme splnili bod o formulaci problému. Druhý bod postupu nám popisuje extrakci dat a přeměnění je na informaci v podobě rovnic či nerovnic. Začněme definováním proměnných. Nechť máme x reprezentující počet krabic typu A a y reprezentující množství krabic typu B. Nyní označíme časovou náročnost na výrobu bedny A jako 2x a náročnost výroby bedny B 3y. Na výrobu máme 1 hodinu neboli 60 minut. Výsledný vztah bude popsán nerovnicí $2x+3y\leq 60$. Dále ze zadání extrahujeme dvě nerovnice popisující podmínku výroby alespoň 5 kusů od obou typů beden. Poslední nerovnici získáme z faktu, že objednávka beden byla nejméně na 25 kusů beden obou typů. Nesmíme zapomenout na důvod řešení této úlohy a tou je maximalizace zisku popsána účelovou funkcí $Z=4x+5y$. Posledním krokem je fakt, že řešíme reálnou úlohu, a proto množství výroby beden musí být rovno či větší 0 stejně tak oba parametry musí být z množiny celých čísel (množina Z). Krátké shrnutí na následujícím obrázku.

$2x+3y\leq 60$	Podmínky:
$x\geq 5$	$x\geq 0$
$y\geq 5$	$y\geq 0$
$x+y\geq 25$	$x\in Z$
$z=4x+5y$	$y\in Z$

Obrázek 14: Podmínky vzorového příkladu bedýnky (vlastní zpracování)

Nyní přejdeme k řešení tohoto modelu. Vzhledem k pouze demonstrační úloze a jednoduchosti příkladu vyřešíme toto zadání pomocí grafického znázornění. Zaneseme všechny nerovnice do grafu. Výsledkem je množina přípustných řešení. Tato množina je ohraničena třemi body a spojnicemi mezi nimi.



Obrázek 15: Vzorový příklad bedýnky grafické řešení [vlastní zpracování]

Požadovaný výsledek v tomto případě je jeden ze tří vrcholů trojúhelníku ohraničující přípustná řešení. Před dosazením parametrů do účelové funkce je potřeba zajistit podmínku o x∈Z, kterou vrchol o souřadnicích x rovné 22,5 a y rovné 5. V tomto případě jsme nuceni vzít nejbližší souřadnici x (hodnotu výroby x), která je celé číslo a splňuje podmínky. Touto souřadnicí x je hodnota 22. Nyní stačí dosadit parametry do rovnice. Z následujícího výpočtu vyplývá, že nejlépe účelová funkce vychází pro x=22 a y= 5.

[x;y]

[15;10] z= 4(15)+5(10)=60+50=110 Kč

[20;5] z= 4(20)+5(5)=105 Kč

[22;5] z= 4(22)+5(5)=113 Kč

Obrázek 16:Vzorový příklad bedýnky výpočet reálného řešení [vlastní zpracování]

Nyní můžeme výsledek interpretovat. Firma by pro maximalizaci svého zisku a za daných podmínek vyrobila 22 kusů beden typu A a 5 kusů beden typu B. Firma na takovéto objednávce maximálně získá 113 Kč čistého zisku.

Posledním bodem je postoptimalizační analýza. Tato analýza se obecně zabývá změnami podmínek po vypočítání úloh lineárního programování. V našem případě by například odpovídala na otázku, jak se změní optimální řešení, jestliže by se změnil například čas objednávky. [36]

3.4.3 Kanonický tvar v lineárním programování

Standardní úloha Lineárního programování může být definována a řešena několika způsoby. Definujme si jí jako maximalizaci nákladové funkce s omezeními a nezáporností pravých stran. Tato definice a její forma zápisu je obecná a lze do ní všechny ostatní problémy lineárního programování přepsat.
Nechť x representuje n-vektor složený z původního zápisu proměnných a jakékoli další proměnné použité k přepisu problému do standardní formy. Standardní úloha lineárního programování se snaží najít **xi, i = 1 až nb** tak, aby maximalizovala účelovou funkci definovanou vztahem.

$$f = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

Obrázek 17: Účelová funkce kanonický tvar [37]

a s množstvím **m** nezávislých podmínek rovnosti

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
\cdot \quad \cdot \quad \cdots \quad \cdot \quad \cdot \\
\cdot \quad \cdot \quad \cdots \quad \cdot \quad \cdot \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m
\end{aligned}
$$

Obrázek 18: Omezující podmínky kanonická forma [37]

a kde $\mathbf{b_i \geq 0,\ i = 1}$ **až** $\mathbf{m}$, a podmínkami nezápornosti na proměnných popsaných vztahem

$$x_j \geq 0;\ j = 1 \text{ to } n$$

Obrázek 19: Podmínky nezápornosti [37]

Parametry b_i ,c_j, a_{ij} nabývají známé hodnoty dle podmínky, kdy $\mathbf{b_i \geq 0,\ c_j,\ a_{ij}}$ **(i = 1** do **m** a **j = 1** do **n)** a kde **m** a **n** jsou celá pozitivní čísla. [37]

3.5 Simplexový algoritmus

Simplexový algoritmus je jedním z nejčastějších způsobů řešení úloh lineárního programování. Algoritmus je iterativní způsob postupu od základního řešení, kdy v každém kroku řešení pozmění proměnné, aby hodnota účelové funkce byla stejná nebo lepší (více optimální) než v předchozí iteraci. Simplexový algoritmus dokáže ovšem vyřešit pouze takové příklady lineárního programování, které v soustavě omezujících podmínek nemají nerovnice typu ”≥”. Dvoufázový simplexový algoritmus tuto problematiku odstraňuje při zachování podmínek pro využití i benefitů standardního simplexového algoritmu. Takže je nutné zajistit kanonický tvar modelu a nezápornost složek vektorů pravých stran. Postup při uplatnění simplexového algoritmu lze rozdělit do několika kroků

1. Příprava
2. Využití metody
3. Interpretace výsledků

Celý tento postup je v každé podkapitole nejdříve popsán a následně aplikován na ukázkovém zadání, které bude provázet celou práci.

3.5.1 Zadání

Firma chce optimalizovat svůj denní výrobní program tak, aby maximalizovala svoje denní tržby. Ty jsou tvořeny tržbami z jednotlivých druhů klenotů. Z každého kusu štrasového náhrdelníku získá firma 5000 Kč, z prstenu 3500 Kč a z každého páru náušnic utrží firma 6000 Kč.

Denní produkce klenotů se ukládá do trezoru, ve kterém je místo celkem na maximálně 30 ks klenotů. Výroba jednoho náhrdelníku spotřebuje 4 hodiny práce zaměstnanců výroby, jednoho prstenu 2 hodiny a výroba jednoho páru náušnic trvá 8 hodin. Celkem je k dispozici 15 zaměstnanců, kteří pracují 8 hodin denně. Na výrobu jednoho náhrdelníku je zapotřebí 1 unce zlata, na výrobu prstenu 0,5 unce a na náušnice se zlato nevyužívá. Každý den má zlatnictví k dispozici maximálně 15 uncí zlata.

3.5.2 Příprava

Před využitím simplexového algoritmu je zapotřebí získání proměnných a vytvoření modelu lineárního programování. Následně je model převeden do kanonického tvaru. Nejdříve definujeme proměnné. Úloha se dotazuje, kolik náhrdelníků, prstenů a náušnic budeme vyrábět. Nechť x1 vyjadřuje množství náhrdelníků, x2 množství prstenů a x3 množství náušnic. Nyní je možné postupně zapisovat rovnice. První omezující podmínka ve tvaru $x1+x2+x3 \leq 30$ popisuje kapacitní podmínku velikosti trezoru. Druhá omezující podmínka opět kapacitního charakteru popisuje časovou náročnost na výrobu jednotlivých klenotů ve vztahu k denní pracovní síle zaměstnanců. Časová náročnost je v textu explicitně zadána a pracovní sílu zaměstnanců vypočteme pomocí množství zaměstnanců a jejich délky pracovní doby. Nerovnice bude ve tvaru $4*x1+2*x2+8*x3 \leq 120$. Poslední omezující podmínka (opět kapacitního charakteru) popisuje spotřebu uncí zlata na jednotlivé šperky k maximální povolené spotřebě. Nerovnice popisující tento vztah je ve tvaru $1*x1+0,5*x2 \leq 15$. Spotřeba na výrobu náušnice je 0 proto v této rovnici proměnná x3 nefiguruje. Tato

úloha požaduje maximalizaci a maximalizační funkce je ve tvaru z=5*x1+3,5*x2+6*x3. Celý model vypadá následovně:

$$x_1 + x_2 + x_3 \leq 30\ (kusy)$$

$$4x_1 + 2x_2 + 8x_3 \leq 120\ (hodiny)$$

$$1x_1 + 0{,}5x_2 \leq 15\ (unce\ zlata)$$

$$z = 5x_1 + 3{,}5x_2 + 6x_3\ (zisk\ v\ tisících)$$

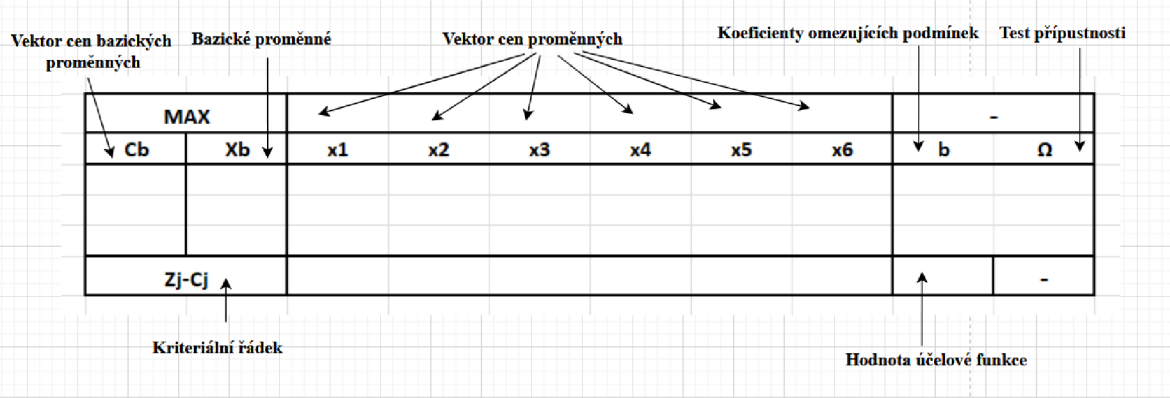
Nyní vyrovnáme případné nerovnice na rovnice. Obecně mohou nastat 3 možnosti. První možností je, že máme nerovnici kapacitního typu (nerovnice typu ≤). V takovém případě nerovnici vyrovnáme přidáním doplňkové proměnné s kladným znaménkem a představuje rezervu. Druhou možností je požadavková podmínka (nerovnice typu ≥). V takovém případě přidáme doplňkovou proměnnou na levou stranu se záporným znaménkem. Poslední možností je podmínka určení (rovnice) a v takovém případě neprovádíme žádnou akci. Doplňková proměnná má v účelové funkci nulovou sazbu a musí splňovat podmínku nezápornosti. Obvykle se takové proměnné pojmenovávají x s dalším nezabraným indexem v řadě, nicméně někteří je pojmenovávají d (doplňkové) s indexem čísla rovnice ve které se nachází.

Nyní převedeme rovnicový tvar na kanonický pomocí přidání pomocných proměnných, abychom zajistili úplnou jednotkovou submatici, dle následujících pravidel. Do rovnic, které vznikly z kapacitních podmínek (nerovnice typu ≤) nepřidáváme pomocnou proměnnou. Do rovnic, které vznikly z požadavkových podmínek nebo podmínek určení přidáme pomocnou proměnnou s kladným znaménkem. Taková proměnná představuje množství jednotek, které zbývají do splnění omezení. Jakékoliv řešení obsahující kladnou hodnotu této proměnné je nepřípustné. Z tohoto důvodu v účelové funkci bude mít prohibitivní sazbu o velikosti alespoň 1 řád vyšší nežli hodnoty ostatních parametrů účelové funkce. Pro pomocné proměnné platí také podmínka nezápornosti. Myšlenka za přidáním pomocných proměnných je zajištění úplné jednotkové submatice. V našem případě pouze kapacitní omezující podmínky, takže žádnou pomocnou proměnnou nepřidáváme. V tento moment je vše připraveno na využití simplexové metody. [36,37,38]

$$x_1 + x_2 + x_3 + x_4 \leq 30\ (kusy)$$
$$4x_1 + 2x_2 + 8x_3 + x_5 \leq 120\ (hodiny)$$
$$1x_1 + 0{,}5x_2 + x_6 \leq 15\ (unce\ zlata)$$
$$z = 5x_1 + 3{,}5x_2 + 6x_3 + 0x_4 + 0x_5 + 0x_6\ (zisk\ v\ tisících)$$

3.5.3 Využití metody

Jsou splněny všechny požadavky na použití Simplexového algoritmu nicméně před jeho aplikací je nutné přepsat model do simplexové tabulky. Na následujícím obrázku je prázdná simplexová tabulka pro 3 rovnice a 6 proměnných. V případě více rovnic by se přidal řádek, v případě 2 rovnic by se jeden odebral. V případě jiného počtu proměnných je změna stejná akorát místo řádků se ubírají nebo přidávají sloupce.



Obrázek 20: Simplexová tabulka prázdná [vlastní zpracování]

Nyní je možné vyplnit parametry do simplexové tabulky: Nejdřív jsou vyplněny ceny jednotlivých proměnných, které jsou rovny příslušnému koeficientu z maximalizační účelové funkce. Následně jsou vyplněny koeficienty parametrů z jednotlivých rovnic a hodnoty pravých stran. Dalším krokem je vyplnění bazických proměnných. Proměnné v bázi splňují, že mají jednotkový vektor.

MAX		5	3,5	6	0	0	0	-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
0	x4	1	1	1	1	0	0	30	
0	x5	4	2	8	0	1	0	120	
0	x6	1	0,5	0	0	0	1	15	
Zj-Cj									-

Obrázek 21: Zlatnictví simplexová tabulka 0 krok. [vlastní zpracování]

Existují případy, kdy zadaná soustava má jednotkovou bázi určenou, nicméně takové případy jsou obvykle teoretického rázu. Pro případy, kdy zadaná optimalizační úloha nebude mít v soustavě omezujících podmínek jednotkovou submatici jsou využity metody umělé báze. Tyto případy jsou popsány v předchozích kapitolách (podmínky ≥ a rovnice) a přidáním pomocných a doplňkových proměnných tento problém eliminujeme. Další z možností je, že soustava omezujících podmínek už je v kanonickém tvaru a opět nebude obsahovat jednotkovou submatici. V takovém případě je přidáno tolik jednotkových umělých vektorů, aby byla vytvořena jednotková báze. Je potřeba si uvědomit, že proměnným doplňkovým jsme přiřadili penalizační koeficienty v účelové funkci, a kvůli své neoptimalitě budou v následujících krocích odstraněny z báze.

Dalším krokem je zjištění, jestli je řešení optimální. K tomu je využit kriteriální řádek. Kriteriální řádek je spočten jako skalární součin cen bazických proměnných a koeficientu příslušné proměnné (Zj) - cena dané proměnné (Cj). Dále je spočtena hodnota účelové funkce jako skalární součin cen bazických proměnných a hodnot pravých stran (koeficienty omezujících podmínek). Nyní je proveden test optima. Test říká, že pro maximalizační (resp. minimalizační) úlohy nesmí kriteriální řádek obsahovat žádné záporné (resp. kladné) hodnoty. Pokud obsahuje, musíme hledat jiné řešení, které má lepší hodnotu účelové funkce. Pokud takové hodnoty neobsahuje je řešení optimální. V našem případě maximalizační úlohy máme hned několik záporných hodnot z čehož vyplývá, že musí být vybrána proměnná, která do báze vstoupí a proměnná, která bázi opustí. Proměnná, která do báze vstupuje je proměnná s nejmenší hodnotou (pro maximalizaci) a největší hodnotou pro (minimalizaci). Tímto je zjištěno, která proměnná vstoupí do báze a získali jsme takzvaný klíčový sloupec (na obrázku označen modře). Nyní k testu přípustnosti. Test přípustnosti určí, která proměnná z báze odejde. Spočteme hodnoty Ω jako podíl hodnot pra-

vých stran a hodnot klíčového sloupce. Dělení je provedeno pouze pro hodnoty klíčového sloupce větší jak 0 neboť takové řádky se klíčovým řádkem stát nemohou. Nyní vybereme minimum z těchto podílů a označíme ho jako klíčový řádek. Pro další kroky je průsečík klíčového řádku a sloupce nazván jako klíčový prvek (někdy známý také jako pivot). Nejnižší podíl je roven 15 a na obrázku je označen žlutou barvou. Klíčový prvek je označen zeleně.

MAX		5	3,5	6	0	0	0	-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
0	x4	1	1	1	1	0	0	30	30
0	x5	4	2	8	0	1	0	120	15
0	x6	1	0,5	0	0	0	1	15	*
Zj-Cj		-5	-3,5	-6	0	0	0	0	-

Obrázek 22: Zlatnictví simplexová tabulka 1 krok. [vlastní zpracování]

Dle předchozího obrázku proměnná x3, která dle testu optimality do báze vstupuje a proměnnou x5, která bázi opouští dle testu přípustnosti. Tím jsme zajistili, jak zlepšení hodnoty funkce, tak zachování podmínek simplexového algoritmu. Nyní aplikujeme Jordanovu eliminační metodu. Nejdříve se zaměříme na klíčový řádek. Klíčový řádek je vydělen hodnotou klíčového prvku, tím je získána na místě klíčového prvku jednička. Zbytek hodnot je opsán po vydělení do nové simplexové tabulky. Nyní je nutné zajistit, aby hodnoty v klíčovém sloupci nad i pod klíčovým prvkem byli rovny nule. Toho je dosaženo tím, že od ostatních původních řádků je odečten vhodný násobek nového klíčového řádku. Tímto postupem je zajištěno neporušení ostatních jednotkových vektorů.

Pro náš případ je postup následovný. Klíčový řádek jsme vydělili 8. Třetí řádek v klíčovém sloupci už je roven nule, a tak ho opíšeme. Pro první řádek je nový klíčový řádek vynásoben -1 a odečten. Dalším krokem je opět test optimality a spočtení hodnoty účelové funkce. Nová simplexová tabulka vypadá následovně.

MAX		5	3,5	6	0	0	0	-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
0	x4	0,5	0,75	0	1	-0,125	0	15	
6	x3	0,5	0,25	1	0	0,125	0	15	
0	x6	1	0,5	0	0	0	1	15	
Zj-Cj		-2	-2	0	0	0,75	0	90	-

Obrázek 23: Zlatnictví simplexová tabulka 1 krok přepočet. [vlastní zpracování]

Vzhledem k faktu, že se v kriteriálním řádku opět nacházejí záporné hodnoty je celý proces opakován. V následujících dvou tabulkách jsou další (v tomto případě 2) kroky k nalezení optimálního řešení.

MAX		5	3,5	6	0	0	0	-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
0	x4	0,5	0,75	0	1	-0,125	0	15	30
6	x3	0,5	0,25	1	0	0,125	0	15	30
0	x6	1	0,5	0	0	0	1	15	15
Zj-Cj		-2	-2	0	0	0,75	0	90	-

Obrázek 24: Zlatnictví simplexová tabulka 2 krok. [vlastní zpracování]

MAX		5	3,5	6	0	0	0	-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
0	x4	0	0,5	0	1	-0,125	-0,5	7,5	15
6	x3	0	0	1	0	0,125	-0,5	7,5	*
5	x1	1	0,5	0	0	0	1	15	30
Zj-Cj		0	-1	0	0	0,75	2	120	-

Obrázek 25: Zlatnictví simplexová tabulka 3 krok. [vlastní zpracování]

MAX		5	3,5	6	0	0	0	-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
3,5	x2	0	1	0	2	-0,25	-1	15	
6	x3	0	0	1	0	0,125	-0,5	7,5	
5	x1	1	0	0	-1	0,125	1,5	7,5	
Zj-Cj		0	0	0	2	0,5	1	135	-

Obrázek 26: Zlatnictví výsledná simplexová tabulka. [vlastní zpracování]

Algoritmus se zastaví ve chvíli, kdy je splněna výše zmíněná podmínka o kriteriálním řádku ať už maximalizace či minimalizace. Mohou nastat speciální případy, kdy se simplexový algoritmus může dostat do cyklu a je záhodno použít Blandovo anticyklické pravidlo. Toto pravidlo mluví o pravidlech vybírání klíčového sloupce dle hodnoty a indexu.

Následně je vyhodnocen výsledek. První možností je neexistence přípustného řešení. Určíme ji dle testu optimality, kdy optimální řešení bylo nalezeno, ale některá z pomocných proměnných zůstala v bázi. Druhou možností je existence právě jednoho optimálního řešení. Jedno řešení nastává v momentě, kdy báze neobsahuje pomocné proměnné a na kriteriálním řádku máme 0 pod bazickými proměnnými a ostatní hodnoty pro kriteriální řádek jsou kladné pro maximalizaci, případně záporné pro minimalizaci. Třetí možností je nekonečně mnoho řešení, kdy řešení je přípustné, pod bazickými proměnnými

v kriteriálním řádku jsou 0 a pod alespoň 1 nebazickou proměnnou je také nulová hodnota. Poslední možností je případ, kdy hodnota účelové funkce může neomezeně růst. Aby nastal tento případ, máme přípustné řešení, v kriteriálním řádku zjistíme, že existují stále proměnné, které zlepší hodnotu účelové funkce nicméně pro ně nemůžeme provést test přípustnosti.

Pro náš případ existuje jen jedno optimální řešení. Toto řešení říká, že se má vyrobit 7,5 kusů náhrdelníků, 15 kusů prstenů a 7,5 kusu náušnic s tržbou 135 tisíc korun. Vzhledem k faktu, že vyrábět půlky prstenu či náhrdelníku není v reálném světě možné musel by být výsledek upraven na nejbližší přípustné řešení, které bude splňovat celočíselnost výroby. Takovým řešením je 7 kusů náhrdelníků, 16 kusů prstenů a 7 kusů náušnic při tržbě 133 tisíc korun.

4 Vlastní práce

Kapitola vlastní práce se zabývá analýzou současného stavu poznání testování simplexových úloh. Zjištěním míst pro optimalizaci a navrhnutí způsobu řešení pro urychlení zadávání testovacích úloh simplexového algoritmu v LMS Moodle. Jednotlivé úskalí zadávaní těchto otázek bude okomentováno. Na základě těchto řešení, jsou formovány požadavky na plugin. Dále tato kapitola popisuje vývoj pluginu a dokumentaci k použití.

4.1 Analýza

4.1.1 Úvod do problematiky

Problematika testování simplexových úloh vychází z obecné problematiky testování znalostí. Takové znalosti lze nejlépe popsat Bloomovou taxonomií konkrétně vzdělávacím okruhem. Obecně lze říci, že obecný test testuje až do třetí úrovně. Tedy testy faktů a jednoduchými interpretacemi aplikací obecné teorie na praxi. Obecné testování simplexového algoritmu se zabývá i čtvrtým stupněm. Různé organizace využívají různého poměru otázek s různou obtížností, nicméně obecný test simplexového algoritmu běžně obsahuje několik teoretických otevřených či uzavřených otázek. Následně největší část testu tvoří otázky na postup výpočtu úlohy simplexového algoritmu a následně post optimalizační úlohy. V následujících podkapitolách jsou jednotlivé typy otázek rozebrány.

4.1.2 Otevřené otázky

Existují 2 typy otevřených otázek. Prvním typem jsou otázky obecně známé pod názvem dlouhé otevřené otázky. V naprosté většině případů se používají pro rozepisovací teoretické otázky, nicméně některými jsou nesprávně používány pro zadávání celých zadání výpočetních úloh. V prvním případě je nemožné zoptimalizovat tento proces vzhledem k faktu, že profesor danou teoretickou úlohu musí zadat a nelze vyhodnotit správnost odpovědi studenta systémem. Více o problému kontroly dlouhého textu viz teoretická část práce současný stav testování. Moodle pro tyto otázky má modul qtype_essay, který přesně tuto funkcionalitu už implementuje a není proto potřeba ji rozšiřovat či předělávat.

Druhým typem otevřených otázek jsou krátké otevřené otázky. Tyto zcela nahrazují nesprávné zadávání výpočetních úloh do dlouhých otevřených otázek. Moodle nabízí

spoustu obecných modulů skládajících se z otázky a velmi krátké odpovědi. Nejčastěji číselné či jednoslovné odpovědi. Bez přídavných modulů komunity to je qtype_shortanswer. Protože je stále po zadávajícím požadován výsledek, vznikly moduly komunity zjednodušující zadávání a generaci náhodných hodnot pro jednotlivé otázky. Asi nejpoužívanějším je qtype_formulas. Více o tomto modulu víz teoretická část. Nicméně žádný takový plugin nepodporuje výpočet hodnot simplexového algoritmu. To nutí zadávající k fixním hodnotám proměnných a vytváření několika sad testů pro studenty, aby každý měl své zadání. Všechny testy musí zadavatel ručně spočítat. Tato část procesu je ideální případ, který lze optimalizovat při zanechání původních vlastností takovýchto otázek.

4.1.3 Uzavřené otázky

Uzavřené otázky jdou také rozdělit do několika skupin podle jejich reálného využití při testování. První možností jsou uzavřené otázky s několika odpověďmi testující teoretické znalosti. Student výběrem jedné nebo více z předem zadaných možností zvolí svoji odpověď, která je systémem vyhodnocena. V této části po zhodnocení není možnost dále optimalizovat, neboť Moodle poskytuje tuto funkcionalitu.

Druhou možností je využívání uzavřených otázek s odpověďmi pro příklady s možnostmi správné odpovědi. Taková otázka předpokládá, že student si spočítá správnou odpověď a zvolí správnou možnost. Nicméně takovéto testování znalostí výpočtu je ekvivalentní s krátkou otevřenou otázkou požadující číselnou odpověď. Jediným případem, kdy takováto otázka nabývá lepších vlastností je v případě například výsledku matice. V takovém případě usnadní studentovi zadávání a lze argumentovat, že odstraňuje potenciální chybu zápisu studenta svého výsledku. Z pohledu optimalizace pro testování simplexového algoritmu je naprostá většina takových uzavřených úloh nahraditelná otevřenou úlohou s krátkou odpovědí a v případě složitějších požadavků na zadání výsledku studentem, lze v naprosté většině úlohu rozdělit do dvou či více částí a tím zjednodušit zadávání výsledku. Je nicméně potřeba si uvědomit, že eliminujeme jednu z vlastností zobrazených řešení studentovi. V případě studentova pochybení, například v jedné hodnotě parametru rovnice a za předpokladu, že žádná z dalších možností není takto špatná, student zjistí, že jeho výsledek je chybný a může se opravit. Takovou vlastnost krátká otevřená možnost neposkytuje.

4.1.4 Požadavky

Z předchozích kapitol je zřejmé, že je potřeba vyvinout modul pro krátkou otevřenou otázku s číselnou odpovědí. Požadavky na řešení krátké otevřené otázky se dají rozdělit do dvou kategorií. První kategorií jsou obecné požadavky pro správnou funkcionalitu otázkového modulu Moodle. Jedná se o strukturu souborů popsané v teoretické části, jejichž kód zabezpečuje od povinných vlastností, jimiž jsou zadání, uložení, zobrazení a další až po metody zajišťující přídavné funkce. Pro tuto práci definici nutných požadavků pro zajištění základní funkcionality otázkového modulu zanedbáme. Nadstavbové požadavky jsou následující:

1. Kategorie
2. Štítky
3. Možnost definování proměnných
4. Možnost stejných hodnot proměnných v několika úlohách

Druhou kategorií jsou požadavky pro simplexový algoritmus. Plugin musí splňovat tyto podmínky:

1. Načtení parametrů účelové funkce
2. Načtení parametrů omezujících podmínek
3. Umožnění určení testované hodnoty
4. Vyhodnocení oproti zadané hodnotě studenta

4.1.5 Obecné požadavky

Zajištění obecných požadavků na funkčnost pro otázkový modul lze řešit třemi způsoby. Prvním komunitou nedoporučený způsob je psaní otázkového modulu takzvaně od nuly. Druhou možností je využití alternativního startu vývoje aneb využití kostry vyrobené Marcusem Greenem. Poslední možností je využití už fungujícího Moodle pluginu a naprogramování funkcí, které pluginu chybí, či úprava stávajících funkcí.

4.1.6 Template / od nuly

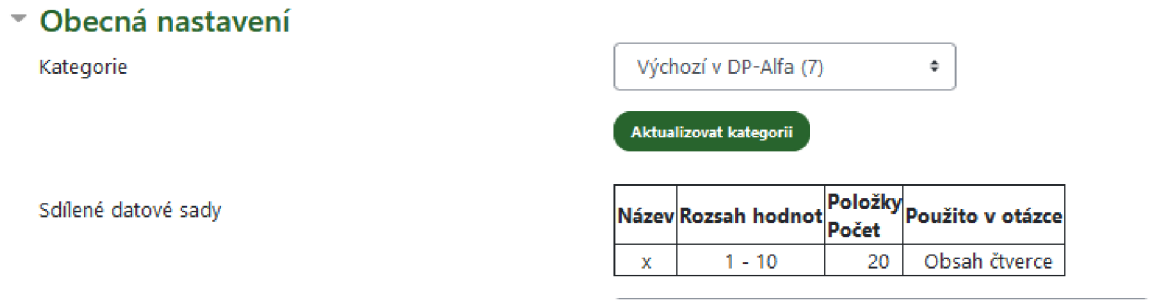
Ačkoliv Moodle je psaný v PHP Moodle má svoje API knihovny zajišťující běžnou funkcionalitu, jakou je získávání dat od uživatele, ukládání či načítání dat z databáze nebo jejich zobrazení. Ačkoliv jsou tyto API knihovny dobře okomentovány, stejně jako question engine 2, to stejné se nedá říct o obecném vývoji otázkových modulů. Při vývoji od nuly je zapotřebí oddědit od správných tříd základní povinnou funkcionalitu, bez které se žádný otázkový modul neobejde. Pro náš projekt bychom například pro definici qtype_název_question potřebovali oddědit od třídy question_graded_automatically_with_countback za předpokladu krátké otázky, která je automaticky vyhodnocena Moodle a s podporou adaptavního režimu(s podporou adaptivního režimu je to ještě trochu složitější). Alternativní cestou k manuálnímu hledání správných metod popsaných nikoliv v dokumentaci, ale komentářích zdrojového kódu je generátor skeletonů Markuse Greena. V tomto generátoru je nutné zvolit typ modulu (qtype_název v našem případě). Dále se generátor zeptá na osobní informace a je potřeba zadat, které soubory chcete, aby byli vytvořeny. Výsledný skeleton je zhruba z 80 procent správný. Skeleton vytvoří správnou strukturu tříd a metod (prázdných), které musí být vyplněny kódem, nicméně už neohlídá oddědění od správných tříd v závislosti na daném typu otázkovového modulu. V našem případě question_graded_automatically pro případ bez podpory adaptivného režimu či už dříve zmíněnou question_graded_automatically_with_countback.

Všechny třídy a jejich metody obstarávající základní funkce typu zobrazení, zadávání, ukládání nebo ověřování správné odpovědi mají svého rodiče se základním fungováním takové otázky. Tyto třídy a metody jsou použité v základních otázkách Moodle typu essay či short answer. V případě jakékoliv deviace od základu je nutné přepsat některou z těchto metod v rámci odvozené třídy pro otázkový modul. Ve výsledku v momentě odchylky od už předem definované otázky je nutné přepsat všechny metody. To je logické, neboť pokud požaduji zadat novou informaci v rámci otázky musím upravit formulář zobrazující zadávání otázky, aby umožnil takovou informaci zadat. Následně je nutné s touto informací pracovat, a přepsat metody pro zpracování, uložení a načtení, stejně jako upravit všechny metody, na které má informace vliv.

4.1.7 Existující plugin

Jiným pohledem na vývoj takového pluginu nabízí využití už existujícího pluginu. Takováto myšlenka je možná jen díky obecné veřejné licenci GNU, pod kterou je všechen kód Moodle licencovaný. Za zmínku stojí říct, že různé části kódu jsou pod verzemi 1-3. Tato myšlenka řešení tedy využívá základního řešení, které implementuje všechny funkcionality. Jak už bylo zmíněno takový plugin, který by přímo podporoval testování simplexového algoritmu neexistuje, proto je nutné najít takový plugin, který svojí funkcionalitou bude co nejblíže požadavkům. Je nutné najít takový plugin, který rozvíjí krátkou otevřenou otázku s automatickým opravováním systému a možností nejen deferred módu, ale případně i módu adaptivního. Možnost zaznamenávání kategorií a štítků a vytváření proměnných a jejich použití.

Kategorie otázek a možnost označení otázek štítky jsou dnes už standardem, a proto najít otázkový modul splňující tyto podmínky není problém. V přístupu k náhodným proměnným existují dva přístupy, které existují. První přístup aplikují například otázky typu calculated (existuje celá rodina calculated otázek), kdy vytvořené globální hodnoty otázky proměnných lze v rámci jedné kategorie použít stejné. Takže při vytvoření nové otázky nabídne uživateli možnost použití globálních proměnných použitých v jiných otázkách v rámci sdíleného setu dat v dané kategorii.



Obrázek 27: Calculated datasets [vlastní zpracování]

Druhým přístupem k problematice sdílení hodnot proměnných je sdružení otázek sdílejících stejná data a tím pádem mající stejné téma do jedné otázky rozdělené na částí. Příkladem takového otázkového typu je Formulas. Formulas sice také rozlišuje globální a lokální proměnné, ovšem ve smyslu globálních proměnných viditelných ve všech částech příkladu a lokálních proměnných viditelných pouze částí, u které jsou definované. Obecná

podoba a funkcionalita otázkového typu Formulas viz teoretická část věnovaná právě tomuto tématu.

4.1.8 Rozhodnutí postupu

Jako nejlepší možnost byl vybrán postup s vývojem nad už zavedený plugin. Důvodů pro toto rozhodnutí je několik. Jak vyplývá z předchozích kapitol, pokud by vývoj začal od nuly po nějakém čase by se dostal do fáze skeletonu a následně by se transformoval do stejného stavu, jako je plugin už vytvořený komunitou. V momentě vývoje vlastního pluginu by vznikl ekvivalentní kód k už existujícímu kódu komunity. Zároveň takovýto kód je roky otestovaný uživateli, a tak by měl být bez chyb a zároveň zůstane zachována návaznost a zkušenosti zadavatelů s úlohami.

Podle průniku zadaných v předchozí kapitole o obecných požadavcích existují 3 otázkové moduly splňující požadavky. Calculated, Stack, Formulas. Modul Stack splňuje podmínky nicméně pro náš vývoj se nehodí, neboť je především vytvořený pro odpovědi formulované rovnicemi. Calculated je velmi starý otázkový modul a dnes už zastaralý. Poslední plugin Formulas je pro náš vývoj nejvhodnější. Formulas splňuje všechny požadavky a zároveň přidává další funkcionality, jako jednotky odpovědí a další.

4.1.9 Způsob napojení

Napojení podpory simplexové funkcionality na modul Formulas může být proveden dvěma způsoby. Prvním způsobem je vytvoření vlastního textového pole či jiného objektu pro zadávání hodnot. Nicméně tento způsob není vhodný z důvodu narušení struktury zadávání pro příklady, které netestují simplexový algoritmus a tím pádem by toto pole či objekt nechávali prázdným. Druhý, elegantnější způsob spočívá ve využití výpočtu funkcí, které Formulas podporuje. Formulas v rámci chování k funkcím rozlišuje mezi konstantami, funkcemi unárními, binárními, speciálními a dalšími typy. Speciální funkce, jako je sčítání listu čísel, práce se řetězci znaků či výběr z množiny čísel. V rámci Formulas není problém takovou speciální funkci přidat a zanechat veškerou funkcionalitu v původním stavu.

4.1.10 Zajištění požadavků na simplexový algoritmus

Simplexový algoritmus se skládá z účelové funkce a omezujících podmínek. Jednotlivé parametry jsou součástí zadání úlohy. Je nutné od zadavatele úlohy získat tyto hodnoty. Uživatel může tyto hodnoty zadat fixně přímo v textu nebo pro náhodné proměnné musí každou proměnnou definovat, ať už pomocí rozsahu náhodných čísel nebo z listu, ze kterého se při vytváření instance otázky vybere hodnota pro proměnnou. Tyto definované proměnné zapíše do textu otázky a při zobrazení otázky se studentovi zobrazí korespondující hodnota.

Simplexový algoritmus má obecně x proměnných v y omezujících podmínkách plus parametry účelové funkce. Tyto podmínky mohou být v nekanonickém tvaru. Tento stav je nutné odstranit jedním ze dvou způsobů. Prvním způsobem je nechat uživatele zadávat omezující podmínky v nekanonickém tvaru. Tento přístup ovšem sebou nese nemožnost volat balanční proměnné v rámci textu. Možnost volání by musela být implementována přes pomocnou funkci vypisující danou proměnnou. Dále by nastávali problémy se strukturou textového zadání. Pro různé počty parametrů omezujících podmínek a proměnných by musel být upraven úvodní text úlohy. Druhou možností, je nechat uživatele zadat omezující podmínky už v kanonickém tvaru. Tím se tento problém vyřeší, nicméně takové řešení vyžaduje úpravu rovnic zadavatelem otázky. Tato úprava trvá maximálně v rámci minut, a proto je přípustným řešením.

4.1.11 Načtení parametrů a serializace

Zadavatel má data v kanonickém tvaru a je připraven je zadat do systému. Systém musí ověřit, jestli zadaná data jsou platná, ve správném množství a zadaná ve správném formátu. Pro tento problém jsou data rozdělena při zadávání do 3 částí.

- Řídící část
- Účelová část
- Omezující část

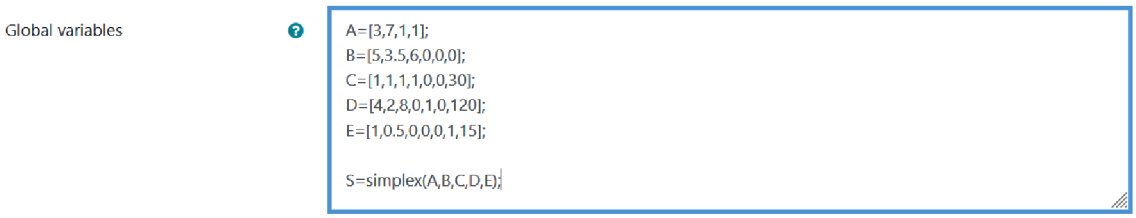
Řídící část se skládá právě ze 4 hodnot určujících počet omezujících rovnic (omezující podmínky v kanonickém tvaru), počtu proměnných, jestli jde o minimalizaci či maxi-

malizaci a čísla určující návratovou hodnotu z tabulky. Počet omezujících rovnic určuje počet sekvencí parametrů, které bude funkce požadovat po zadavateli. Počet proměnných určuje počet proměnných v každé sekvekvenci parametrů. Do tohoto čísla se započítává i strana b rovnice. Další hodnota určuje, jestli jde o minimalizační nebo maximalizační účelovou funkci. K tomuto bodu je nutné si uvědomit, že proměnné, které byli zadavatelem přidány by měli mít správnou hodnotu parametru v závislosti na typu účelové funkce a typu (pomocná nebo doplňková proměnná). Poslední hodnota určuje návratovou hodnotu z předem sestavené tabulky hodnot. O této tabulce v samostatné kapitole. Tato první část je ohraničena hranatými závorkami a jednotlivé části jsou odděleny čárkami.

Účelová část je druhým vstupním parametrem funkce a obsahuje seznam parametrů účelové funkce. Jsou seřazeny zleva do prava dle indexu. Jsou opět v hranatých závorkách a jednotlivé hodnoty jsou odděleny od sebe čárkami.

Poslední částí je omezující část. Tato část se opakuje dle hodnoty v řídící části počtu rovnic. Hodnoty jednotlivých parametrů jsou v hranatých závorkách a odděleny čárkou a jsou seřazeny dle indexu zleva doprava.

Po definici jednotlivých částí je potřeba si definovat oddělovač mezi jednotlivými částmi. Jednotlivé části jsou od sebe rozděleny čárkou. Celá funkce i s volacím názvem bude tedy například vypadat následovně. Jsou využity pomocné proměnné A-E. Toto volání funkce simplex je vytvořeno pro modelové zadání z teoretické části.



Obrázek 28: Návrh syntaxu [vlastní zpracování]

Nyní je nutné ověřit správnost zadaných dat. Konkrétně množství zadaných parametrů do funkce simplex, množství parametrů v rámci řídící části, množství proměnných, správně zadaných typu účelové funkce, množství parametrů omezujících podmínek či nezápornosti hodnot stran b.

Dále je potřeba zkontrolovat, jestli zadaný index hodnoty z tabulky odpovídá možné odpovědi z tabulky a jestli je pro danou soustavu rovnic taková hodnota spočitatelná. Poslední kontrolou je přítomnost jednotkové matice v zadaných datech.

4.1.12 Vyhodnocení a vrácení výsledku

Systém načetl hodnoty parametrů všech částí, ověřil jejich správnost a našel jednotkovou matici. Třída provede příslušný počet kroků k nalezení optimálního řešení. Následně porovná zadavatelem zadaný index oproti tabulce a zavolá příslušnou hodnotu nebo metodu pro získání požadované hodnoty.

Ačkoliv zadáním matice parametrů omezujících podmínek je pořadí rovnic v řádcích určeno, tak při studentově průchodu testem a sestavování podmínek může jednotlivé omezující podmínky prohodit. V takovém případě, pokud by zadavatel vyplňoval fixní indexy hodnot z tabulky, mohl by systém studentovu odpověď porovnávat oproti špatné hodnotě. Je tedy žádoucí v rámci volání hodnot ustanovit jiný systém.

Při vytváření dostupných návratových hodnot je potřeba zohlednit otázky, které jsou studentům pokládány. Prvním typem otázek jsou otázky na správné převedení do kanonického tvaru. Tyto otázky nemusí plugin implementovat vzhledem k faktu, že je zadávající musí vyplnit a tím pádem je může testovat bez knihovny. Druhou částí jsou otázky na postup řešení do nejlepšího řešení. Jak vyplývá z teorie je nutné najít, která proměnná vstupuje do báze, která vystupuje a následně přepočítat parametry celé tabulky. Tento proces se opakuje do nalezení nejlepšího řešení. Tyto otázky obvykle směřují jen na průběžnou kontrolu studenta, kdy v případě chyby v některém z kroků je možné přiřadit alespoň nějaké body. Další nejrozšířenější skupinou otázek jsou otázky směřující na výslednou simplexovou tabulku. Poslední skupinou jsou postoptimalizační otázky.

4.1.13 Návratové indexy a hodnoty

Index může nabývat kladných i záporných hodnot. Každá číslice v číslu je platný znak včetně záporného znaménka. Index je složen z číslic unikátně určující návratovou hodnotu. Pro naprostou většinu indexů platí notace x pro sloupec proměnné (sloupec x1-x9) a pro označení řádku y (opět x1-x9). V některých indexech jsou fixní hodnoty či mínusové znaménko pro vytvoření většího množství unikátních kombinací použitelných pro indexy.

4.1.13.1 Hodnota účelové funkce

Nejlepší hodnota účelové funkce je jednou z nejčastějších otázek. Pro navrácení této hodnoty byl zvolen index 0 nebo 100.

4.1.13.2 Hodnota řádku zj-cj

Hodnoty řádku zj-cj pro výslednou simplexovou tabulku mají index 10,20,30,40…90 se syntaxí x0 (číslo proměnné, 0). Pro proměnnou x1 je hodnota indexu zj-cj rovná 10. Na následujícím obrázku jsou zobrazeny hodnoty, které takto lze vypsat.

MAX								-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
	x1	11	21	31	41	51	61	1	
	x2	12	22	32	42	52	62	2	
	x3	13	23	33	43	53	63	3	
Zj-Cj		10	20	30	40	50	60	0	-

Obrázek 29: Simplexová tabulka řádek zj-cj [vlastní zpracování]

4.1.13.3 Průsečíky tabulky

Hodnoty vnitřku tabulky mohou být na různém místě v závislosti na studentově případném nedodržení optimálního postupu či vytvořením omezujících podmínek v jiném pořadí, než byli zadány v textu úlohy. Další návazný problém nastává při náhodné generaci čísel pro simplexový algoritmus. Může nastat situace, že se budeme ptát na pole tabulky (průsečík proměnné x a bazické proměnné y), která neexistuje. Nejelegantnějším způsobem řešení tohoto problému je možnost vrácení jména proměnné z báze a tím umožnit zadavateli vyřešit tento problém.

Syntax indexu pro průsečíky tabulky jsou následovné: xy, kde x je jméno indexu proměnné a y jméno indexu bazické proměnné. Složením těchto dvou indexů nám vznikne index popisující přesné pole v simplexové tabulce. Pro existující pole vrací hodnotu pro neexistující pole vrací 0. Krajní hodnoty tohoto indexu jsou 11-99 včetně.

MAX								-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
	x1	11	21	31	41	51	61	1	
	x3	13	23	33	43	53	63	3	
	x2	12	22	32	42	52	62	2	
Zj-Cj		10	20	30	40	50	60	0	-

Obrázek 30: Simplexová tabulka průsečíky [vlastní zpracování]

4.1.13.4 Hodnoty b stran

Index pro návrat hodnoty b pro proměnnou x. Jediný problém, který může nastat je, že zadaná proměnná není v bázi při optimálním výsledku. V takovém případě je návratová hodnota 0. V jakémkoliv jiném případě je navrácena hodnota b pro proměnnou.

Pro návrat hodnot byly vyhrazeny indexy 1-9. 1 pro x1 ,2 pro x2 …

MAX								-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
	x1	11	21	31	41	51	61	1	
	x3	13	23	33	43	53	63	3	
	x2	12	22	32	42	52	62	2	
Zj-Cj		10	20	30	40	50	60	0	-

Obrázek 31: Simplexová tabulka hodnota b [vlastní zpracování]

4.1.13.5 Maximální možná hodnota b pro proměnnou

Tyto indexy se zaměřují na navrácení hodnoty omega pro zadanou proměnnou. Je navrácena nejoptimálnější z hodnot. Této hodnoty je dosaženo stejným způsobem, jako kdybychom chtěli zjistit při neoptimálním řešení, která proměnná vystoupí z báze. V tomto případě je klíčový sloupec vybrán zadavatelem a klíčový řádek je vybrán systémem. Umožňuje tedy otázky typu: Na jaké maximální úrovni lze do řešení zařadit proměnnou x1?

Pro návrat hodnot byly vyhrazeny indexy -1 až -9. Pro x1 -1, x2 -2 …

4.1.13.6 Hodnota pivota

Index pro návrat hodnoty pivota v xtém kroku postupu výpočtu simplexového algoritmu. Ačkoliv umístění této hodnoty se může měnit v závislosti na pořadí sestavení jednotlivých podmínek studentem, jeho hodnota je vždy stejná. Dále hodnota ověřuje správné vybrání klíčového sloupce a klíčového řádku.

Pro návrat hodnot byly vyhrazeny indexy -11 až -19. Pro x1 -11, x2 -12 …

4.1.13.7 Jméno bazické proměnné

Pomocný index. Někdy je vhodné si zajistit, že proměnná, na kterou se zadavatel bude ptát je bazická. Tyto návratové indexy umožní zadavateli zadat do ostatních výstupních indexů takovou sekvenci, aby nedocházelo k nulovým odpovědím či odpovědím mimo bazická řešení.

Pro návrat hodnot byly vyhrazeny indexy -21 až -29. Pro název bazické proměnné na prvním řádku -21 druhé proměnné na druhém řádku -22 a tak dále.

MAX								-	
Cb	Xb	x1	x2	x3	x4	x5	x6	b	Ω
	-21	11	21	31	41	51	61	1	
	-22	13	23	33	43	53	63	3	
	-23	12	22	32	42	52	62	2	
Zj-Cj		10	20	30	40	50	60	0	-

Obrázek 32: Simplexová tabulka jméno bazické proměnné [vlastní zpracování]

4.1.14 Návrh tříd

4.1.14.1 Qtype_formulas_simplex_table

Třída qtype_formulas_simplex_table (jméno dle notace Moodle) je třída jehož instance v sobě uchovávají všechny informace o simplexové tabulce. Třída tedy bude obsahovat struktury na uložení parametrů účelové funkce a hodnot parametrů omezujících podmínek rozdělených na hodnoty levých a pravých stran. Dále bude poskytovat struktury pro uložení aktuální báze, stejně jako strukturu pro uložení hodnot řádku zj-cj. Třída dále bude obsahovat parametr určující další krok postupu simplexového algoritmu a strukturu na uložení jednotlivých hodnot pivotů po jednotlivých krocích algoritmu. Pro všechny hodnoty bude obsahovat sety a gety.

4.1.14.2 Qtype_formulas_simplex

Třída qtype_formulas_simplex bude obsahovat struktury na uložení vstupních dat od zadavatele. Dále bude obsahovat strukturu na uložení počáteční báze a jednu instanci třídy qtype_formulas_simplex_table. Třída bude obsahovat metody pro ověření možnosti navrácení požadovaného indexu. Dále bude obsahovat metody pro spočtení kroků simplexového

algoritmu a bude si jednotlivé mezivýsledky ukládat do instance třídy qtype_formulas_simplex_table. Třída také vlastní metody pro navrácení požadovaného výsledku zadavatelem.

4.2 Implementace

Následující kapitoly se zabývají samotnou implementací nadstavby pluginu Formulas. Pro vývoj byl použit lokální testovací server Moodle a plugin Formulas ve verzi 4.91, který v této verzi podporuje Moodle 3.0 až 3.9. Moodle i Formulas byly nastaveny s anglickou lokalizací, a i nadstavba je pro lokalizaci v anglickém jazyce. Pro psaní a úpravu kódu byl využit PhpStorm ve verzi 2021.3.

4.2.1 Kontrola vstupů

V této části kódu je zabezpečena validnost vstupů funkce simplex. Vyhodnocení globálních a lokálních proměnných proběhlo a systém načetl textové pole vyplněné zadavatelem úlohy. Formulas rozparsoval zápis funkce (simplex(A,B,C,D,E)) z textu, kde nahradil proměnné seznamy definovaných čísel dle části proměnné. Formulas rozpozná funkci simplex a jednotlivé argumenty funkce rozdělené čárkou si uloží do listu hodnot $values a počet těchto parametrů v $sz.

Nejdříve je ověřen minimální počet vstupujících argumentů, které zadavatel zadal. Minimální počet 4 se skládá z jednoho řídícího, účelového a alespoň 2 argumentů pro podmínky. Stejně jako je zkontrolován počet argumentů oproti hodnotě zadané zadavatelem v řídícím argumentu. Dále je zkontrolována hodnota, jestli se jedná o maximalizaci či minimalizaci a množství hodnot účelové funkce.

```
case 'simplex': //[number of equations,number of variables,min/max
function,what i want to return],[parameters of min/max
function],[parameters of equations],[...]....
    if (! ($sz >= 4 && $values[0][0]==$sz-2)) { //at least 1 control
parameters +1 max/min parameter + 2 equations parameters to make sense
        throw new
Exception(get_string('error_invalid_number_of_simplex_parameters',
'qtype_formulas'));
    }
    if (!( count($values[0])==4 )) { //first parameter has to have 4
control values inside
        throw new
Exception(get_string('error_invalid_number_of_controlling_parameters',
'qtype_formulas'));
    }
    if (!( $values[0][1]>=2 && $values[0][1]>=$values[0][0]) ) { // at
least 2 variables but more equations need at least same amount of
variables as equations
        throw new
Exception(get_string('error_invalid_number_of_variables',
'qtype_formulas'));
    }
    if (!( $values[0][2]==0 || $values[0][2]==1) ) { //min(0) or max(1)
function
        throw new Exception(get_string('error_invalid_min_max',
'qtype_formulas'));
    }
    if (!( count($values[1])==($values[0][1]-1)) ) { //check if
zparameters have right length
        throw new Exception(get_string('error_invalid_array_size',
'qtype_formulas', 2));
    }
```

Kód 1: Kontrola vstupů 1. část

V další části je ověřeno správné množství parametrů omezujících podmínek a kladná hod-

nota pravých stran těchto podmínek.

```
for ($i = 2; $i <$sz; $i++) { //check for cparamaters arrays
    if (!( count($values[$i])==($values[0][1])) ) { //check if all
parameters of equations have right length +1 for b value
        throw new Exception(get_string('error_invalid_array_size',
'qtype_formulas', $i+1));
    }
    if (!($i >= 2 && $values[$i][$values[0][1]-1]>0 ) ) { //check
if parameters b are positive numbers
        throw new
Exception(get_string('error_invalid_value_of_b_side',
'qtype_formulas', $i+1));
    }
}
```

Kód 2: Kontrola vstupů 2. část

V poslední části jsou všechny proměnné poslány do třídy qtype_formulas_simplex a je provedena kontrola jednotkové matice, a jestli je možné požadovaný index odpovědi vrátit. V poslední části je string (simplex(A,B,C,D,E)) nahrazen číselnou hodnotou a je uložen zpět.

```
$parameters=array();
 for ($i = 2; $i <$sz; $i++) {
        array_push($parameters, $values[$i]);
 }
$scalculator=new
qtype_formulas_simplex($values[0],$values[1],$parameters);
 if (!($scalculator->unit_matrix()) ) { //exist basis
     throw new
Exception(get_string('error_invalid_parameters_does_not_contain
_unit_matrix', 'qtype_formulas'));
 }
 if (!($scalculator-
>possible_index_precalculate($values[0][3])) ) { //check if i
can provide such answer {check for out of bound variable etc}
     throw new
Exception(get_string('error_invalid_parameter_of_answer',
'qtype_formulas'));
 }
 $scalculator->calculate();
 $this->replace_middle($vstack, $expression, $l, $r, 'n',
$scalculator->return_answer_index($values[0][3]));
```

Kód 3: Kontrola vstupů 3. část

4.2.2 Třída qtype_formulas_simplex_table

Instance této třídy slouží k uložení základního řešení a následně případnou úpravou

```
class qtype_formulas_simplex_table {
    protected $rowsTable;//number of y
    protected $callsTable; //number of x
    protected $zparameters = [];
    protected $table = []; //values cparameters-b values
    protected $basis = [];
    protected $B = []; //values of b
    protected $delta = []; //zj-cj
    protected $pivot = [];
    protected $final = ""; //need to be recalculated
```

Kód 4: Proměnné qtype_formulas_simplex_table

hodnot instance v závislosti na nutnosti přepočítání. Tato třída obsahuje pouze gety a sety jednotlivých proměnných a funkce obsluhující inicializaci a hromadný update hodnot. Tato třída rozděluje vstupní data do příhodnějších struktur a přidává struktury pomocných proměnných.

Konkrétně jde o listy hodnot zj-cj ($delta), proměnných, které jsou v bázi a string určující, jestli je nalezeno nejlepší řešení nebo je potřeba dalších kroků k nalezení optimálního řešení. Třída obsahuje také list $pivot pamatující si hodnoty jednotlivých pivotů v rámci postupu výpočtu.

4.2.3 Třída qtype_formulas_simplex

Třída dle návrhu obsahuje počáteční hodnoty poskytnuté zadavatelem ve formě stringů. Dále obsahuje strukturu pro ukládání počáteční báze a proměnnou $table, do které bude uložena instance třídy qtype_formulas_simplex_table. Ta je následně pomocí metod přepisována směrem k optimálnímu řešení. V následujících podkapitolách jsou tyto metody, stejně jako metody kontrolující vstupní parametry a návratové metody popsány.

```
class qtype_formulas_simplex {
    protected $control; // control data
    protected $zparameters; //original data
    protected $cparameters; // original data
    protected $basis = [];// original basis
    protected $table; //table evolving data
```

Kód 5: qtype_formulas_simplex definice proměnných

4.2.3.1 Inicializace

Klasická metoda obsluhující inicializaci instance třídy. Vstupními argumenty jsou data zadané zadavatelem. Kontrolní prvek je uložen přímo. Do účelové funkce je přidán na konec pole označení minimalizace či maximalizace z kontrolního prvku a do omezujících podmínek je přidáno rovnítko.

```
public function __construct(array $control , array $zparameters, array
$cparameters)
{
    $this->control=$control;
    $this->zparameters = $zparameters;
    if ($control[2]==1)
    {
        array_push($this->zparameters,'max');
    }else{
        array_push($this->zparameters,'min');
    }
    for($i = 0; $i < count($cparameters); $i++){
        $temp = end($cparameters[$i]);
        array_pop($cparameters[$i]);
        array_push($cparameters[$i],'=',$temp);
    }
    $this->cparameters=$cparameters;
```

Kód 6: qtype_formulas_simplex inicializace

4.2.3.2 Unit_matrix

Metoda unit_matrix slouží k ověření, jestli hodnoty parametrů omezujících podmínek obsahují jednotkovou matici. Metoda projde všechny sloupce a v momentě co najde sloupec, který by mohl být součástí jednotkové matice si ho uloží do jednotkové matice a odnastaví hodnotu z $limitationIndex na příslušném místě. Po projití všech sloupců hodnot parametrů metoda zhodnotí, jestli bylo dosaženo celé jednotkové matice a v takovém případě vrátí true. V opačném případě vrátí false.

```
public function unit_matrix()
{
    $limitationIndex = range(1, (count($this->cparameters)));
    $unitColumn = false;
    for ($i = 0; $i < count($this->zparameters)-1 ; $i++) {
        $numberOfZeros = 0;
        $possibleLimitation = -1;
        for ($j = 0; $j < count($this->cparameters); $j++) {
            if ($this->cparameters[$j][$i] == 1) {
                $unitColumn = true;
                $possibleLimitation = $j;
                continue;
            }
            if ($this->cparameters[$j][$i] == 0) {
                $numberOfZeros++;
            } else
                $unitColumn = false;
        }
        if ($unitColumn === true && $numberOfZeros == count($this-
>cparameters) - 1) {
            array_push($this->basis, $i);
            unset($limitationIndex[$possibleLimitation]);
        }
    }
    if( count($limitationIndex)==0){
        return true;
    }else{
        return false;
    }
}
```

Kód 7: qtype_formulas_simplex unit_matrix

4.2.3.3 Possible_index_precalculate

Metoda possible_index_precalculate přijímá $index (hodnotu indexu zadaného zadavatelem) a vrací bool hodnotu určující, jestli je daný index v intervalu povolených hodnot indexů a následně v některých případech ověří, jestli pro jednotlivé cifry indexu je možné za zadaných hodnot parametrů účelové funkce a parametrů omezujících podmínek spočítat hodnotu výsledku. Příkladem může být dotaz na hodnotu zj-cj řádku pro proměnnou x8, když v zadaném modelu je pouze 7 proměnných. Tato metoda je volána z třídy qtype_formulas_variables za ověřením obecných vlastností parametrů.

```
    $countc = count($this->cparameters) - 1;
    $countz = count($this->zparameters) - 1;
    switch ($index) {
        case ( $index == 100||$index == 0):// z
            return true;
        case ($index >= 1 && $index <= 9): //values of b side
            if($index>$countz){
                return false;
            }else return true;
        case ($index >= 11 && $index <= 99 && fmod($index, 10) !=
0): //table body value
            $i = str_split($index);
            if($i[0]>$countz){
                return false;
            }else return true;
        case ($index >= 10 && $index <= 99 && fmod($index, 10) ==
0): // zj-cj
            $i = str_split($index);
            if($i[0]>$countz){
                return false;
            }else return true;
        case ($index >= -9 && $index <= -1): // max possible b of
variable
            if(abs($index)>$countz){
                return false;
            }else return true;
        case ($index >= -19 && $index <= -11): //pivot value
            $i = str_split($index);
            if($i[2]-1>count($this->cparameters) ){
                return false;
            }else return true;
        case ($index >= -29 && $index <= -21 ): //name of basis
            $i = str_split($index);
            if($i[1]>$countc){
                return false;
            }else return true;
        default:
            return false;
```

Kód 8: : qtype_formulas_simplex possible_index_precalculate

4.2.3.4 Reverse_zparameters

Metoda reverse_zparameters řeší problém s minimalizací. V případě minimalizace by museli být na všechny výpočty napsány dvě metody, které by v závislosti na minimalizaci či maximalizaci úlohy aplikovali postup simplexového algoritmu. Tento problém jde ovšem vyřešit pomocí pravidla zmíněného v teoretické části. Pokud zadavatel zadal minimalizaci můžeme jí řešit jako maximalizaci, pokud vynásobíme parametry účelové funkce -1. Nesmí být ovšem zapomenuto na opětovné vynásobení parametrů účelové funkce a také vynásobení řádku zj-cj mínus jedničkou po provedení všech kroků simplexového

algoritmu nebo prezentování výsledků. V našem případě se tak děje výlučne v moment navrácení výsledků v metodě return_answer_index.

Do metody vstupují parametry účelové funkce a návratová hodnota je znegovaná účelová funkce.

```
protected function reverse_zparameters($zparameters)
{
    for ($i=0;$i<count($zparameters)-1;$i++){
        $zparameters[$i] = $zparameters[$i] * (-1);
    }
    return $zparameters;
}
```

Kód 9: qtype_formulas_simplex reverse_zparameters

4.2.3.5 Create_table

Metoda create_table ukládá do proměnné $this->table instanci třídy qtype_formulas_simplex_table s počátečními hodnotami. Metoda v případě minimalizace zneguje vstupní parametry účelové funkce.

```
protected function createTable()
{
    if (end($this->zparameters) === 'max'){
        $this->table = new qtype_formulas_simplex_table($this-
>zparameters, $this->cparameters, $this->basis);
    }else {
        $this->table = new qtype_formulas_simplex_table($this-
>reverse_zparameters($this->zparameters), $this->cparameters, $this-
>basis);
    }
}
```

Kód 10: qtype_formulas_simplex create_table

4.2.3.6 Calculate

Metoda calculate je metoda, která zajišťuje průběh mezi úvodní simplexovou tabulkou a tabulkou finální. Metoda nejdříve zavolá inicializaci úvodní tabulky. Následně v každém kroku nechá spočítat, jestli je nutný další krok simplexového algoritmu nebo není možné pokračovat. V případě nutnosti dalšího kroku nechá tabulku přepočítat (hodnota parametru recalculation), v opačném kroku přeruší cyklus(hodnota parametru decided). Může nastat případ kdy není možné provést přepočet simplexové tabulky. Podrobněji popsáno v podkapitole metody recalculation.

```
public function calculate()
{
    $this->createTable();
    do {
        $this->decision();
        if ($this->table->getFinal() === 'recalculation') {
            $this->recalculation();
            if($this->table->getFinal() === 'impossible') {
                break;
            }
        } else {
            break;
        }
    } while (1);
}
```

Kód 11: qtype_formulas_simplex calculate

4.2.3.7 Decision

Metoda Decision je metoda rozhodující o následujícím kroku simplexového algoritmu. Pokud je nutný přepočet tabulky k dosažení lepší hodnoty účelové funkce nastaví parametr $final instance třídy qtype_formulas_simplex_table na recalculation. V opačném případě nastaví tento parametr na decided. V průběhu metody a vyhodnocování hodnot nutných ke správnému postupu tyto pomocné hodnoty ukládá do objektu.

Metoda začíná vynulováním parametru určující přepočet a vytvořením dvou pomocných proměnných jedna na uložení řádku zj-cj($delta) a druhou na uložení jedné hodnoty ze stejného řádku. Metoda pro každý sloupec spočítá hodnotu a uloží si jí do připravené proměnné. (Více o postupu výpočtu viz teoretická část)

```
protected function decision()
{
    $this->table->setFinal("");
    $delta = [];
    $tmpDelta = 0;
    // count zj-cj line
    for ($i = 0; $i < $this->table->getRowsTable(); $i++) {

        for ($j = 0; $j < $this->table->getCallsTable(); $j++) {
            $tmpDelta += $this->table->getzparameters()[$this->table-
>getBasis()[$j]] * $this->table->getTable()[$j][$i];
        }
        $tmpDelta -= $this->table->getzparameters()[$i];
        array_push($delta, $tmpDelta);
        $tmpDelta = 0;
    }
```

Kód 12: qtype_formulas_simplex decision část 1

V další části kódu je přidána hodnota účelové funkce na konec struktury. Metoda následně updatuje hodnoty řádku zj-cj$ (delta).

```
// Value of z
for ($j = 0; $j < $this->table->getCallsTable(); $j++) {
    $tmpDelta += $this->table->getzparameters()[$this->table-
>getBasis()[$j]] * $this->table->getB()[$j];
}
array_push($delta, $tmpDelta);
$this->table->setDelta($delta);
```

Kód 13: qtype_formulas_simplex decision část 2

Poslední částí je kontrola zápornosti hodnot řádku zj-cj. Foreach prochází hodnoty do té doby, dokud nenajde první zápornou hodnotu. Následně uloží verdikt o následujícím postupu simplexového algoritmu.

```
    // is delta negative set recalculation
    foreach ( $this->table->getDelta() as $value) {
        if ($value == end($delta))
            continue;
        if ($value < 0) {
            $this->table->setFinal('recalculation');
            break;
        }
    }
    if(!($this->table->getFinal()==='recalculation')){
        $this->table->setFinal('decided');
    }
}
```

Kód 14: qtype_formulas_simplex decision část 3

4.2.3.8 Recalculation

Metoda Recalculation je metoda, která přepočítá simplexovou tabulku s neoptimální hodnotou simplexového algoritmu do tabulky s lepší hodnotou účelové funkce. Metoda nejdřív najde nejmenší hodnotu řádku zj-cj (klíčový sloupec) následně určí proměnnou vystupující z báze (klíčový sloupec). Následně s nalezeným pivotem přepočítá simplexovou tabulku.

V první části kódu metody jsou definované pomocné proměnné a je nalezena minimální hodnota řádku zj-cj. Tímto je jednoznačně určen klíčový řádek.

```
protected function recalculation()
{
    $newX = 0;
    $replacementX = 0;
    $replacementXMatrix = [];
    $table = $this->table->getTable();
    $B = $this->table->getB();
    $basis = $this->table->getBasis();

    // lowest number in delta
    $min = 1000000;
    foreach ($this->table->getDelta() as $key => $value) {
        if ($value == $this->table->getDelta()[count($this->table-
>getDelta())-1]) {
            continue;
        } else {
            if ($value < $min) {
                $newX = $key;
                $min = $value;
            }
        }
    }
```

Kód 15: qtype_formulas_simplex recalculation část 1

V druhé části kódu je nalezen klíčový řádek (proměnná opouštějící bázi) a je zkontrolována možnost, že není možné přepočítat simplexovou tabulku kvůli 0 nebo negativním hodnotám omegy pro všechny řádky. Při průchodu je uložena nejmenší kladná nenulová hodnota omegy pro další kroky.

```
// which x will get from base
for ($i = 0; $i < $this->table->getCallsTable(); $i++) {

    if ($this->table->getTable()[$i][$newX] <= 0) {
        array_push($replacementXMatrix, '-');
        continue;
    } else
        array_push($replacementXMatrix, $this->table->getB()[$i] / $this-
>table->getTable()[$i][$newX]);
}

$impossible = true;
// can be recalculated?
$min = 1000000;
foreach ($replacementXMatrix as $key => $value) {
    if ($value !== '-')
    {
        $impossible = false;
        if ($value < $min) {
            $replacementX = $key;
            $min = $value;
        }
    }
}
```

Kód 16: qtype_formulas_simplex recalculation část 2

Poslední část kódu přepočítává hodnoty jednotlivých řádků simplexové tabulky. Nejdříve si spočítá hodnoty klíčového řádku za pomocí hodnoty pivota a následně pomocí klíčového řádku přepočítá ostatní řádky. Na konci metoda updatuje základní data

```
if($impossible) {
    $this->table->setFinal('impossible');
}else {
    // recount of table
    $basis[$replacementX] = $newX;
    $tmp = $table[$replacementX][$newX];
    $this->table->setPivotValue($table[$replacementX][$newX]);

    for ($i = 0; $i < $this->table->getRowsTable(); $i++) {
        $table[$replacementX][$i] /= $tmp;
    }
    $B[$replacementX] /= $tmp;

    for ($i = 0; $i < $this->table->getCallsTable(); $i++) {
        $tmpRow = [];
        if ($replacementX === $i)
            continue;
        $tmp = $table[$i][$newX];

        for ($j = 0; $j < $this->table->getRowsTable(); $j++) {
            array_push($tmpRow, $table[$replacementX][$j] * - ($tmp));
        }

        for ($j = 0; $j < $this->table->getRowsTable(); $j++) {
            $table[$i][$j] += $tmpRow[$j];
        }
        $B[$i] += - ($tmp) * $B[$replacementX];
        unset($tmpRow);
    }
    $this->table->updateTable($table, $basis, $B);
}
```

Kód 17: qtype_formulas_simplex recalculation část 3

4.2.3.9 Return_answer_index

Metoda return_answer_index je metoda obstarávající návratovou hodnotu dle indexu. V momentě, kdy je tato metoda volána jsou provedeny všechny kontroly vstupů a jsou hotové všechny výpočty. Dle hodnoty indexu a typu odpovědi je návratová hodnota vrácena buď z tabulky spočtených hodnot optimálního řešení, nebo je proveden výpočet a následně je hodnota navrácena. Pokud se jednalo o minimalizaci je nutné u některých výstupních hodnot jejich hodnotu znegovat.

První část kódu popisuje návrat hodnoty pro indexy označující hodnotu účelové funkce a hodnotu pravé strany b pro zadanou proměnnou.

```
public function return_answer_index($index)
{
    switch ($index) {
        case  ($index == 100||$index == 0 ):  //z
            if (end($this->zparameters) === 'max'){
                return $this->table->getDelta()[count($this->table-
>getDelta()) - 1];
            }else {
                return $this->table->getDelta()[count($this->table-
>getDelta()) - 1]*-1;
            }
        case ($index >= 1 && $index <= 9): //values of b side
            for($i = 0; $i <count($this->table->getBasis()); $i++){
                if ($index-1==$this->table->getBasis()[$i]){
                    return $this->table->getB()[$i];
                }
            }
            return 0;
```

Kód 18: qtype_formulas_simplex return_answer_index část 1

Druhá část kódu popisuje navrácení hodnot z těla tabulky, hodnoty zj-cj řádku pro zadanou proměnnou a hodnotu nejnižší omegy pro zadaný index klíčového sloupce.

```
case ($index >= 10 && $index <= 99 && fmod($index, 10) != 0): //table
body value
    $i = str_split($index);
    for($a=0;$a<$this->table->getcallsTable();$a++){
        if($this->table->getBasis()[$a]+1==$i[1]){
            return $this->table->getTable()[$a][$i[0]-1];
        }
    }
    return "0";
case ($index >= 10 && $index <= 99 && fmod($index, 10) == 0): // zj-cj
    return $this->table->getDelta()[($index/10)-1];
case ($index >=-9 && $index <= -1 ): // max possible b of variable
    $lowest=10000000;
    for($i = 0; $i <count($this->table->getB()); $i++) {
        $possiblelow=$this->table->getB()[$i]*$this->table-
>getTable()[$i][abs($index)-1];
        if (($lowest>$possiblelow)&&($possiblelow>0)){
            $lowest=$possiblelow;
        }
    }
    if ($lowest==10000000){
        return 0;
    }else{
        return $lowest;
    }
```

Kód 19: qtype_formulas_simplex return_answer_index část 2

Poslední část kódu má na starost výpis hodnoty pivota pro požadovaný krok postupu a výpis názvu báze ve zvoleném řádku.

```
case ($index >= -19 && $index <= -11 ): //pivot value
        $i = str_split($index);
        return $this->table->getPivot()[$i[2]-1];
    case ($index >= -29 && $index <= -21 ): //name of basis
        $i = str_split($index);
        return $this->table->getBasis()[$i[2]-1]+1;
}
```

Kód 20: qtype_formulas_simplex return_answer_index část 3

4.2.4 Přidání dalších indexů

Pro přidání dalších návratových hodnot je postup následovný. Je nutné si vyhradit rozmezí hodnot nepřiřazené jiným návratovým indexům. Následně je nutné přidat case v metodě possible_index_precalculate, kdy pro požadované indexy bude case vracet true. Dalším krokem je přidání case případu do metody return_answer_index. V rámci tohoto případu je nutné napsat svou logiku nového indexu. Posledním krokem je zvednutí verze o jedna v souboru version.php a restartování serveru.

5 Výsledky a diskuse

5.1 Testování

Testování lze rozdělit na dvě části. První částí je testování základního pluginu Formulas s nadstavbou. Pro toto testování byl využit oficiální postup vydaný Moodlem k testování funkčnosti nových otázkových typů. Druhou částí je testování funkčnosti nadstavby.

5.1.1 Testování funkčnosti pluginu

Testování funkčnosti základního pluginu Formulas proběhlo testy, které jsou v základu součástí pluginu a následně byl proveden oficiální Moodle postup pro ověřování funkčnosti nových modulů. Tento postup složený z jedenácti bodů ověřuje kompatibilitu s Moodle. Postupně bylo ověřeno vytváření, editování, vyplňování, preview mód a další body postupu. Pro žádný z bodů postupu nebyla nalezena chyba.

5.1.2 Testování funkčnosti nadstavby

Všechny metody byly otestovány jednotkovými testy. Následně byly provedeny testy postupu a výstupů oproti volně dostupným online kalkulačkám. Takovýto test byl proveden pro 10 náhodně vygenerovaných zadání a nebyla nalezena odlišnost mezi postupy ani výsledky. Následně byly otestovány jednotlivé indexy a jejich výstupy oproti spočteným hodnotám online kalkulačky.

5.2 Základy využití

Pro využití výsledku této práce je nutné splnit následující požadavky. Obecné podmínky pluginu Formulas ve verzi 4.91. Těmi jsou Moodle ve verzi v rozmezí 3.0 až 3.9. Dále je nutné nainstalovat adaptivní mód od Tima Hunta nazvaný qbehaviour_adaptivemultipart volně dostupný na adrese https://moodle.org/plugins/view.php?plugin=qbehaviour_adaptivemultipart ve verzi vyšší jak 3.3.

V následující tabulce se nachází obecný zrychlený výtah dostupných indexů. Je doporučeno si přečíst podrobnější popis těchto indexů a jejich návratových hodnot v podkapitolách kapitoly nazvané návratové hodnoty a indexy. Tato tabulka slouží pouze k vizuálnímu zobrazení. Je doporučeno před vytvářením si přečíst podrobnější popis metod. Dále je doporučeno si projít přílohy práce se dvěma vzorovými příklady využití.

Schema	Index	Popis
-	100 nebo 0	Nejlepší hodnota účelové funkce
x0	10,20,30,40...90	Hodnota řádku zj-cj pro proměnnou x.
xy	11-19,21-29,31-39...91-99	Jednotlivé průsečíky tabulky. V případě, že zadaná kombinace nemá průsečík bez přepočtu vrací 0.
x	1,2,3,4...9	Hodnota b pro proměnnou x. Když není v bázi tak 0.
-x	-1.-2.-3.-4.-5...-9	Maximalní velikost b pro proměnnou x
-1x	-11,-12,-13...-19	Vrací název bazické proměnné pro řádek x
-2x	-21,-22,-23,-24...-29	Vrací hodnotu pivota pro x krok výpočtu.

Obrázek 33: Indexy rychlá referenční tabulka [vlastní zpracování]

5.3 Základní požadavky na uživatele

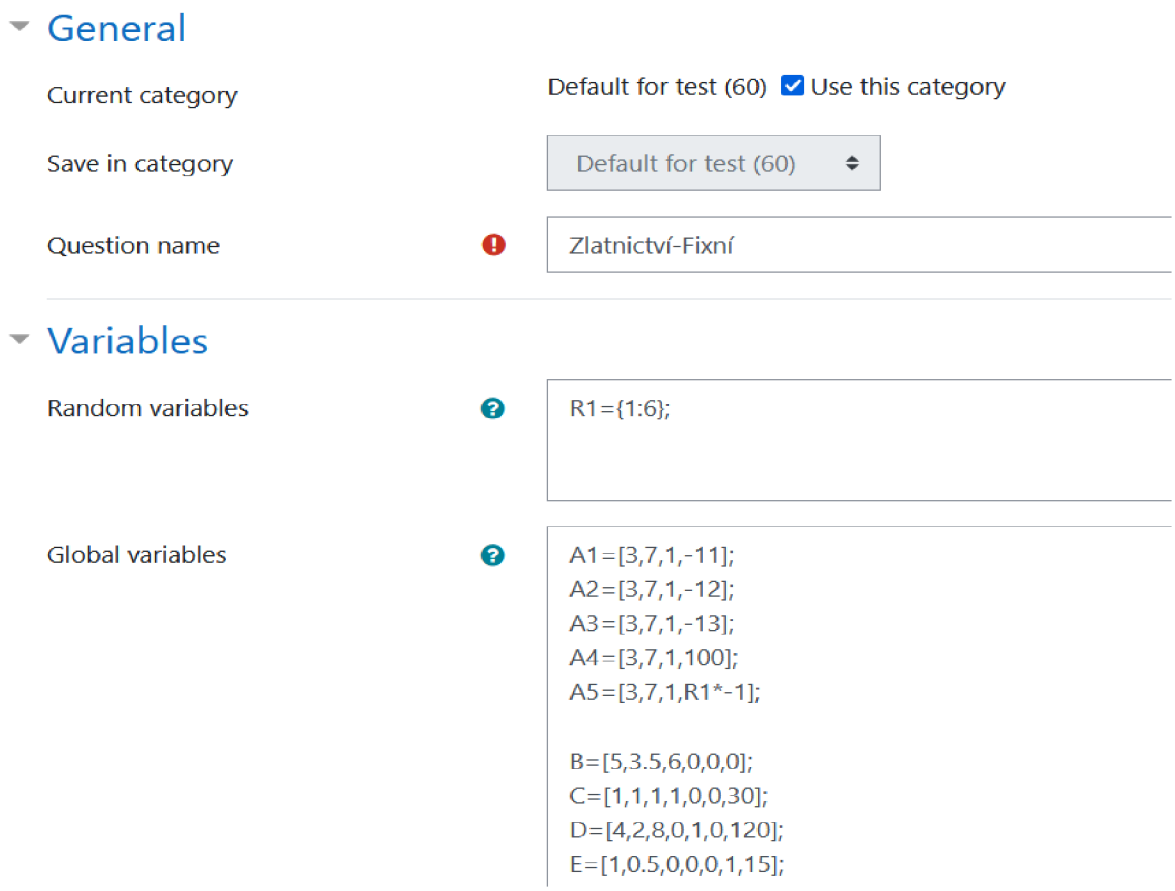
Požadavky na uživatele je nutno rozdělit na požadavky na zadavatele úlohy a požadavky na studenta. Základní požadavky na zadavatele jsou schopnost práce s LMS Moodle. Schopnost vytváření testů a správy otázek v testu. Nadstavbou nutnou pro maximální využití výsledku této práce je schopnost zadavatele pracovat s otázkovým typem formulas. Dokumentace k obecnému použití viz web https://moodleformulas.org. Posledním požadavkem je znalost simplexového algoritmu a seznámení se s dokumentací k použití a seznamem dostupných návratových hodnot. Pro uživatele neznalého simplexového algoritmu je nicméně možné testovou úlohu sestavit dle příkladů použití v této práci. Nicméně není předpokladem, aby testové úlohy na simplexový algoritmus byly vytvářeny zadavatelem neovládající simplexový algoritmus.

Pro studenta se oproti jakékoli jiné klasické otázce nic nemění. Tedy požadavky zůstávají u schopnosti vyplňovat textová pole zobrazena systémem.

5.4 Příklad pro fixní hodnoty

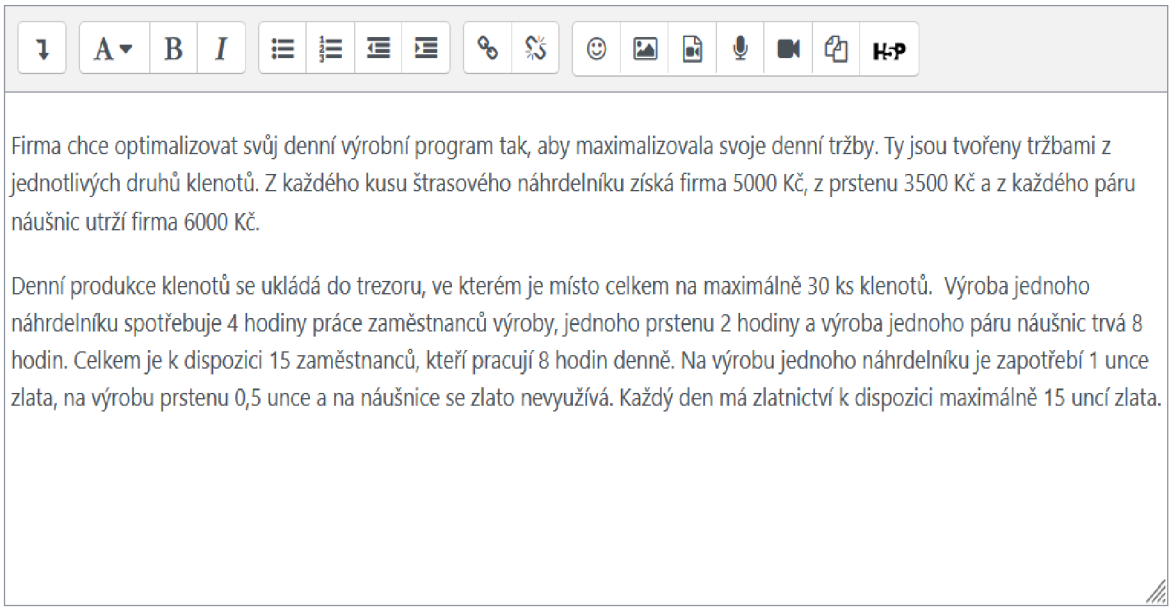
V této podkapitole je zobrazeno použití otázky formulas s nadstavbou funkce simplex. Nejdříve bude zobrazeno zadávání úlohy po částech a následně výsledné zobrazení studentovi. Pro funkční případ maximalizační úlohy je použit příklad z teoretické části s fixními hodnotami.

V první fázi zadávání úlohy je nutné vytvořit proměnné a jejich hodnoty v globálních proměnných. V případě, že se budeme chtít ptát na různé proměnné v podotázkách (pro každého studenta otázka na jinou proměnnou) musíme si vytvořit proměnné generující náhodné hodnoty. V tomto případě byla vytvořena proměnná R1, která při studentově spuštění testu vybere jednu z hodnot. Taková hodnota může být zadána rozmezím nebo seznamem čísel. V tomto ukázkovém příkladu je výběr z rozsahu 1-6 včetně. Následně jsou vytvořeny kontrolní argumenty pro funkci simplex A1 až A5 s rozdílným indexem požadované odpovědi. Je možné si už v této části definovat celou funkci simplexu pod proměnnou a je možné s ní pracovat. V tomto příkladu je ovšem složení až v jednotlivé části podúlohy. Za zmínku stojí využití náhodné proměnné v objektu A5, kdy bude zadavatelem požadován návrat maximální hodnoty pro proměnnou x[R1], kdyby měla vstoupit do báze.



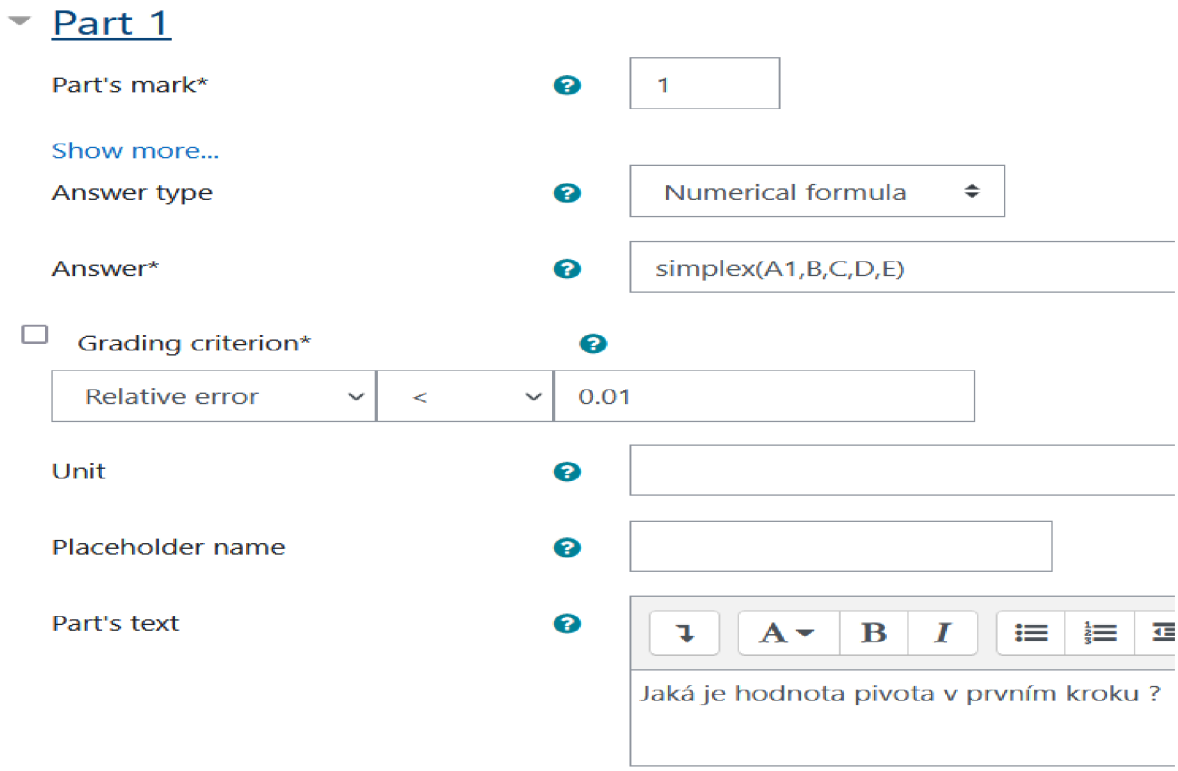
Obrázek 34: Zlatnictví fixní Moodle část 1 [vlastní zpracování]

Ve druhé části je nutné tyto fixní hodnoty pro studenta zobrazit v textu. V tomto případě jsou hodnoty parametrů zapsány jako fixní hodnoty, ale je možné jednotlivé hodnoty všech parametrů pojmenovat a vytvořit jako samostatnou proměnnou a poté zobrazovat. V takovém případě je nutný speciální zápis proměnné v textu. Tento způsob je popsaný v následující kapitole pro náhodné hodnoty, proto je v této části zobrazen tento fixní způsob. Nicméně je potřeba mít na paměti, že použitím toho způsobu v případě změny hodnoty nějakého parametru je nutné tuto hodnotu změnit nejenom v textu, ale také v definici proměnných.

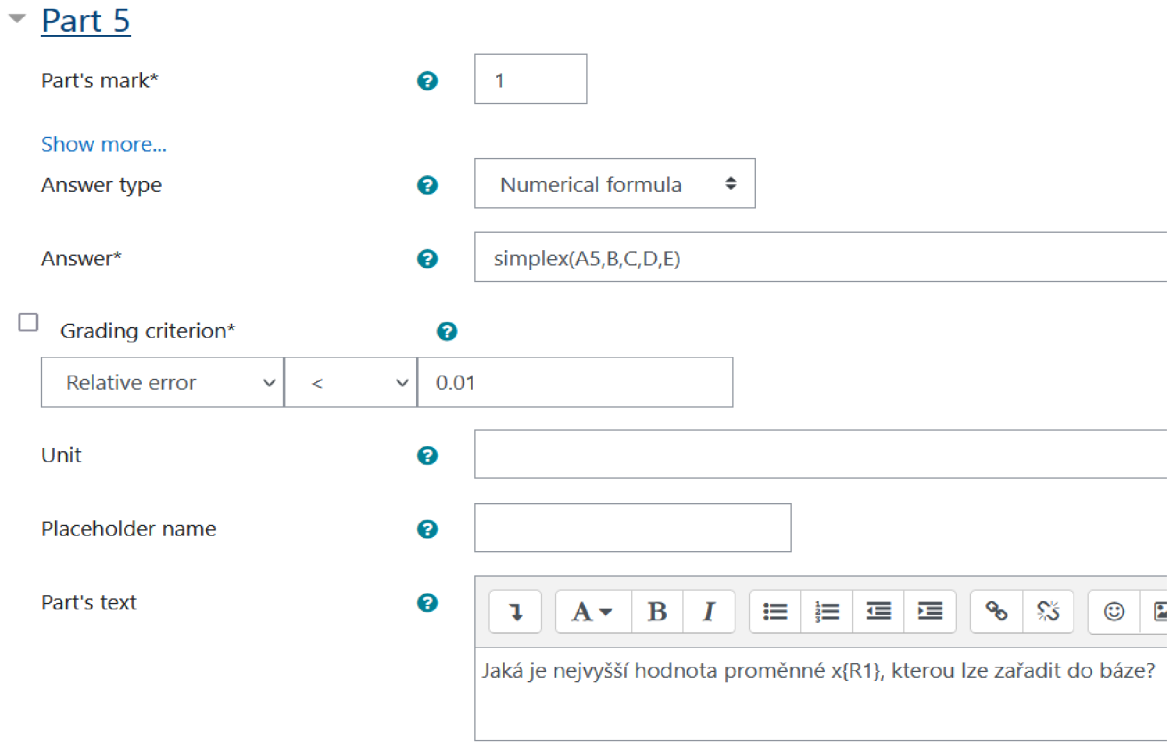


Obrázek 35: Zlatnictví fixní Moodle část 2 [vlastní zpracování]

Poslední část, kterou zadavatel musí vyplnit, jsou výsledky jednotlivých podúloh, jejich bodové ohodnocení a jejich textové vyjádření. Dále je možné nastavit povolenou chybu nebo vytvářet lokální proměnné. Více o obecné funkčnosti formulas viz teoretická část nebo dokumentace formulas. Pro naši úlohu zadáme výsledek pomocí funkce se správným indexem. Pro tuto ukázku je povolená chyba ponechána na defaultní hodnotě. Na následující obrázcích je možné vidět první část s otázkou na hodnotu pivota v prvním kroku simplexového algoritmu a následně poslední část návrat maximální hodnoty pro proměnnou x[R1], kdyby měla vstoupit do báze.

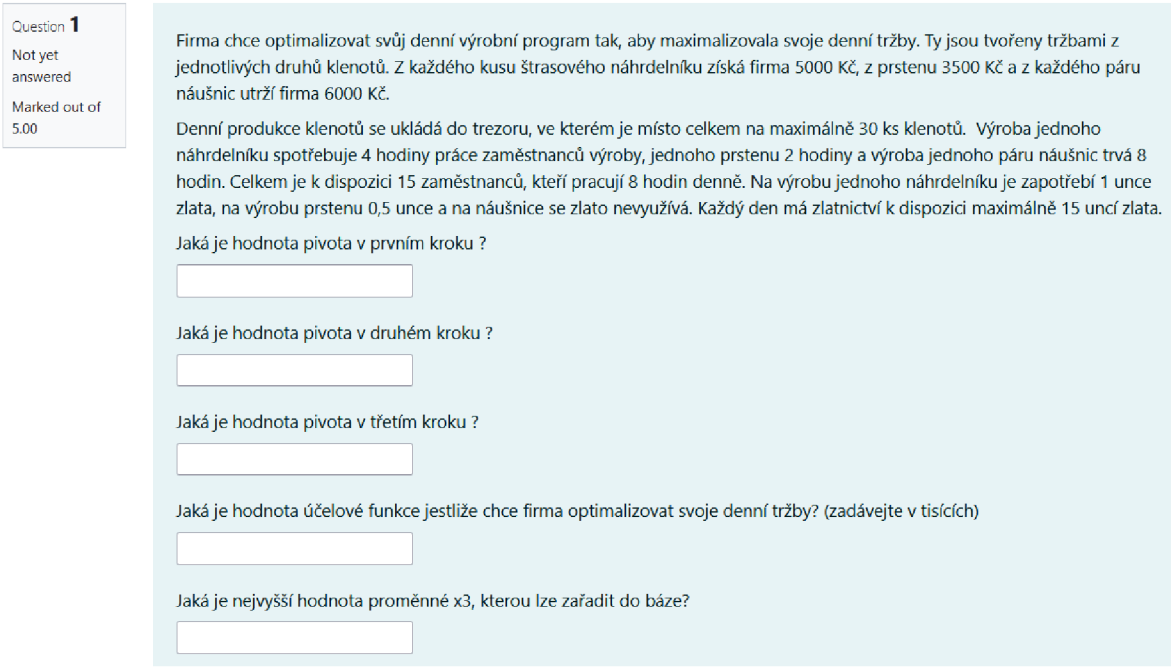


Obrázek 36Zlatnictví fixní Moodle část 3 [vlastní zpracování]



Obrázek 37:Zlatnictví fixní Moodle část 4 [vlastní zpracování]

V tomto momentě lze otázku uložit a zařadit do testu. V momentě, co student test otevře se mu zobrazí úloha následovně. V našem případě jsme všem podotázkám nechali hodnocení 1 bodu za správnou odpověď, proto celková otázka má hodnotu 5 bodů. Pro toto spuštění byla náhodně vybrána proměnná x3.



Obrázek 38:Zlatnictví fixní Moodle část 5 [vlastní zpracování]

5.5 Příklad pro náhodné hodnoty

V této podkapitole je zobrazeno použití otázky formulas s nadstavbou funkce simplex. Nejdříve bude zobrazeno zadávání úlohy po částech a následně výsledné zobrazení studentovi. Pro funkční případ maximalizační úlohy je použitý příklad z teoretické části s náhodně generovanými hodnotami.

Výsledný plugin podporuje náhodnost hodnot parametrů, nikoliv náhodnost modelu. Model nemění počet omezujících podmínek ani počet parametrů v nich. Ačkoliv je možné všechny proměnné včetně jejich počtu nechat vygenerovat, museli bychom také upravovat dle vygenerovaných čísel text úvodu úlohy. Tento postup, ačkoliv možný, není plánova-

ným způsobem využití nadstavby. V plánovaném případě je zadána fixní „kostra“ příkladu a mění se jen parametry neovlivňující strukturu úlohy. Nemění se počet pomocných či doplňkových proměnných stejně jako celkový počet proměnných účelové funkce nebo omezujících podmínek. V některých případech se nemění ani nějaké parametry jednotlivých rovnic.

Pro ukázkový případ opět využijeme příklad zlatnictví. V první části je nutné vygenerovat veškeré proměnné parametrů, které budeme potřebovat a neurčili jsme je, jako statické. Doporučený postup je nejdříve definovat globální struktury s vyplněním míst určených pro náhodnou generaci čísel proměnnou. V našem případě nejdříve definujeme řídící parametry pro budoucí podotázky (Zpracování proměnných probíhá v pořadí zápisu, není tedy možné volat nevytvořenou proměnnou). Následně definujeme hodnoty parametrů účelové funkce. V našem případě 3 proměnné jsou náhodné a 3 jsou statické, neboť to jsou proměnné pomocné a jejich změna by ovlivnila strukturu a tím pádem text úlohy. Dalším krokem je vytvoření struktury popisující omezující podmínky. Jednotková matice zůstává fixní, stejně jako v tomto případě první omezující podmínka. Důvod je vyplývající ze zadání. Ostatní hodnoty jsou nahrazeny názvy náhodných proměnných. Za zmínku stojí proměnná D7, která označuje v zadání celkový počet hodin dostupných pro práci na špercích. V zadání je práce zaměstnance definována, jako 8hodinová doba krát generované množství zaměstnanců. Bylo by možné tuto dobu také náhodně měnit, ale to záleží na osobní preferenci zadavatele úlohy.

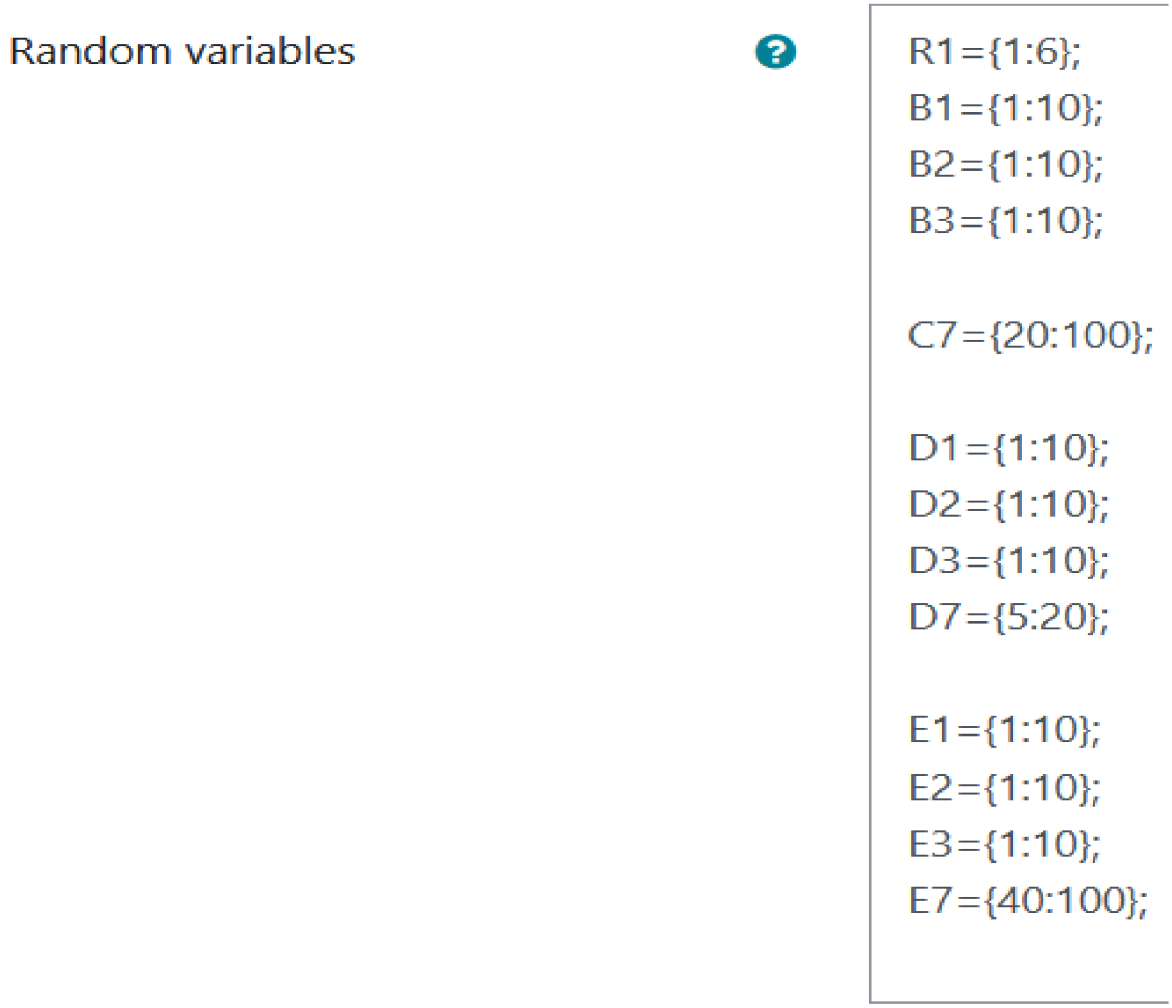
Global variables

```
A1=[3,7,1,-11];
A2=[3,7,1,-12];
A3=[3,7,1,-13];
A4=[3,7,1,100];
A5=[3,7,1,R1*-1];

B=[B1,B2,B3,0,0,0];
C=[1,1,1,1,0,0,C7];
D=[D1,D2,D3,0,1,0,D7*8];
E=[E1,E2,E3,0,0,1,E7];
```

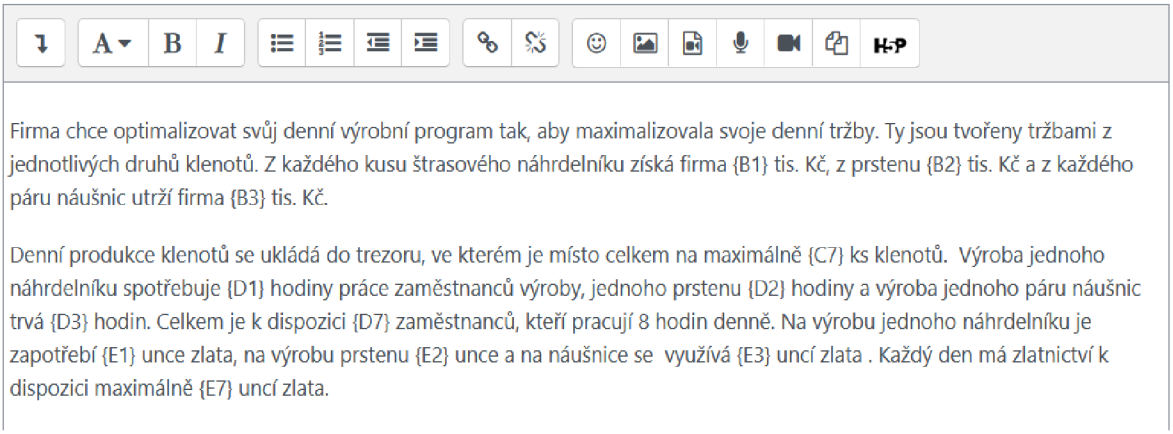
Obrázek 39: Zlatnictví náhodné Moodle část 1 [vlastní zpracování]

Nyní je nutné pro všechny proměnné, které jsme navrhli v globálních proměnných, vytvořit proměnné generující jejich náhodnou hodnotu. Jak už bylo zmíněno, je možné nastavit generaci z rozsahu nebo ze seznamu čísel či textů. V ukázkovém případě byly použity rozsahy. Rozsahy byly definovány v „rozumných“ velikostech.



Obrázek 40: Zlatnictví náhodné Moodle část 2 [vlastní zpracování]

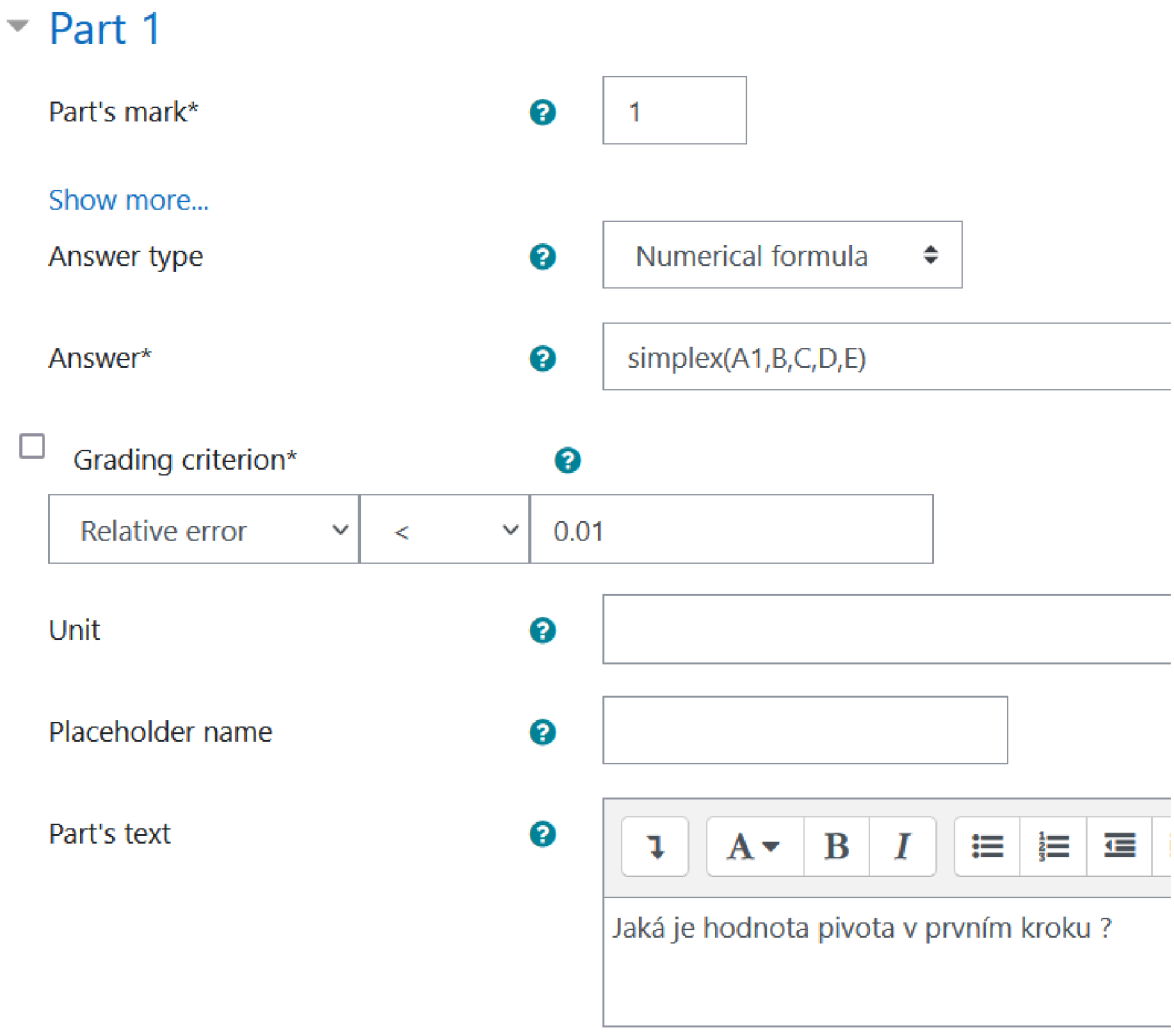
Proměnné jsou definovány a je možné přejít k úvodnímu zadání. V zadání je nutné nahradit fixní hodnoty proměnných hodnotami námi generovaných proměnných. Při vytváření úplně nové úlohy, a tedy i úvodního textu je doporučeno postupovat následovně. Napsat úvodní text s proměnnými, které budou mít požadovanou strukturu zadání. Následně vytvoření globálních proměnných a konkrétní generovaná čísla. V případě neobsahující jednotkové matice či jiného syntaxového problému vytváření proměnných či simplexové syntaxe je zobrazen při pokusu o uložení error navádějící na chybné místo. Pro náš úvodní text zlatnictví by úvodní text vypadal následovně.



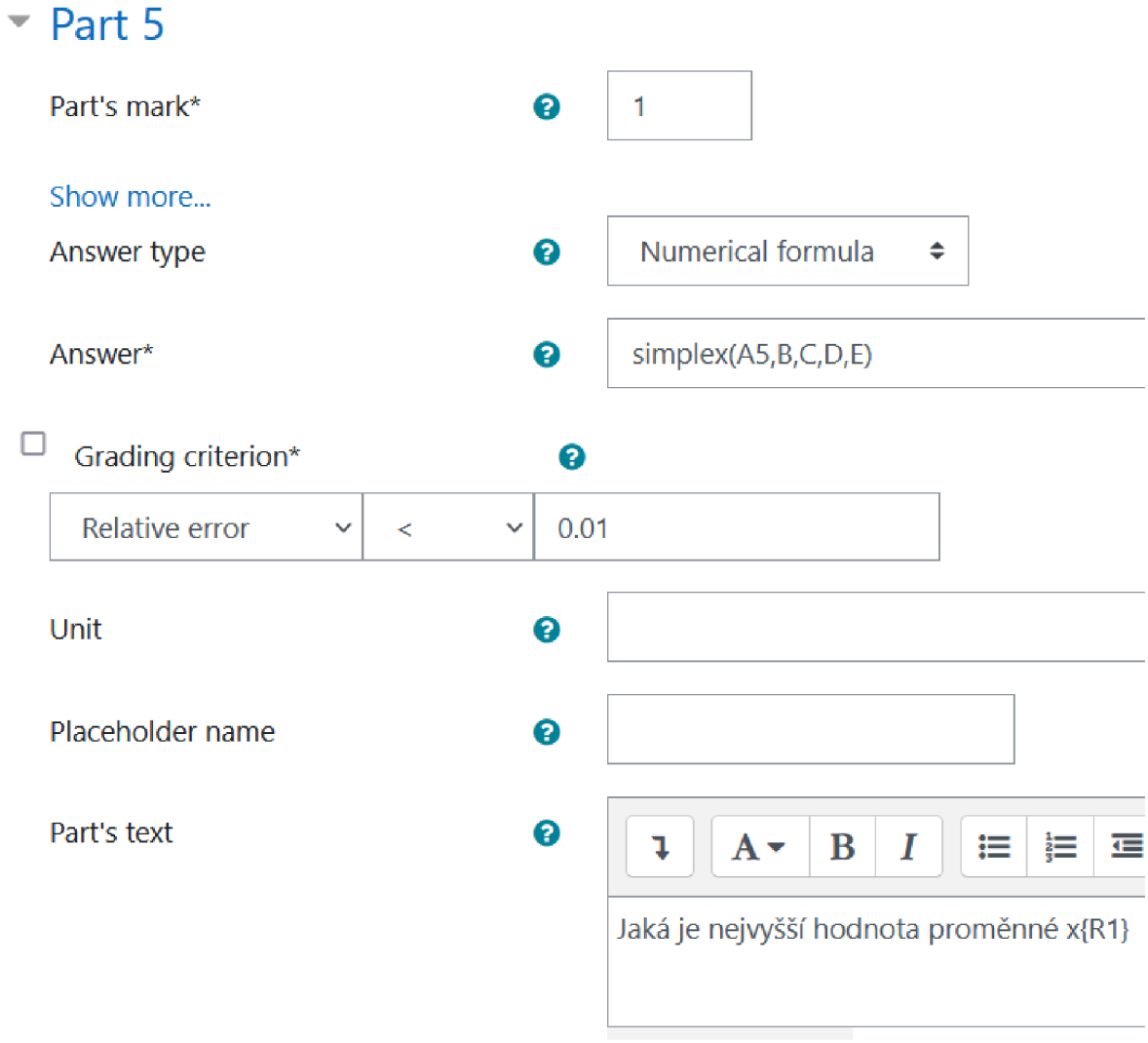
Obrázek 41: Zlatnictví náhodné Moodle část 3 [vlastní zpracování]

Poslední částí, která se od fixního zadávání neliší, je zadávaní jednotlivých podotázek a jejich nastavení. Opět je zobrazena první otázka na hodnotu pivota v prvním kroku a následně poslední část návrat maximální hodnoty pro proměnnou x[R1], kdyby měla vstoupit do báze.

.

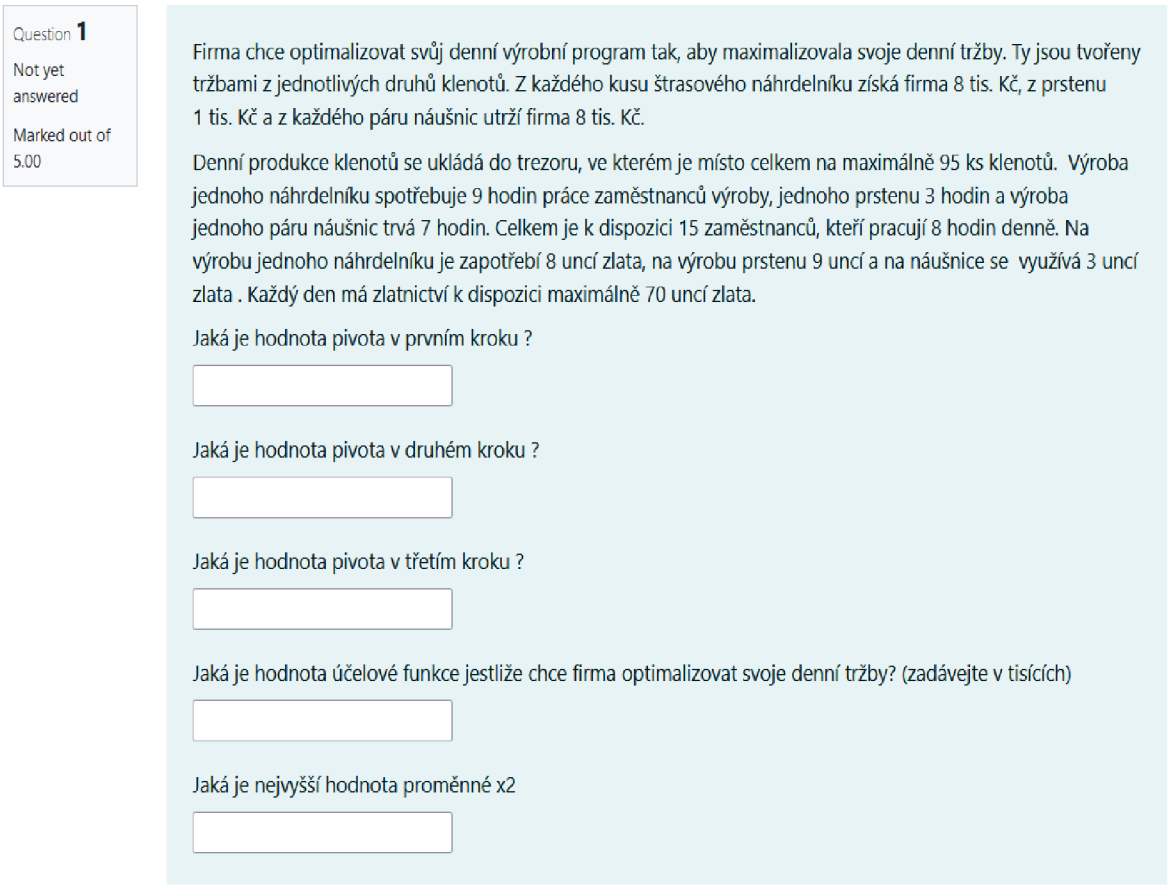


Obrázek 42: Zlatnictví náhodné Moodle část 4 [vlastní zpracování]



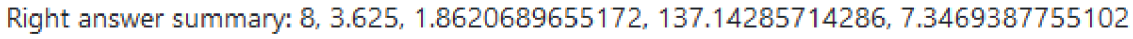
Obrázek 43: Zlatnictví náhodné Moodle část 5 [vlastní zpracování]

V tomto momentě lze otázku uložit a zařadit do testu. V okamžiku, kdy student test spustí, zobrazí se mu úloha následovně. V našem případě jsme všem podotázkám nechali hodnocení 1 bodu za správnou odpověď, proto celková otázka má hodnotu 5 bodů.



Obrázek 44: Zlatnictví náhodné Moodle část 6 [vlastní zpracování]

Pro příklad s těmito vygenerovanými hodnotami jsou odpovědi spočtené knihovnou následující. Studentovi stačí zadat hodnotu v rozsahu povolených hodnot zadavatelem. Rozsah jde nastavit buď relativně k výsledku nebo absolutně, kdy je nutné zadat obě hranice hodnot. V tomto ukázkovém příkladu je ponechána defaultní hodnota. Pro náhodně generované testy je doporučené vždy rozsahy nastavit vlastní. V důsledku náhodné generace je pravděpodobné, že ne všechny úpravy budou v celých číslech a student bude nucen zaokrouhlovat a tím si zanese do svého výsledku chybu.



Obrázek 45: Zlatnictví náhodné Moodle část 7 správné odpovědi [vlastní zpracování]

V některých případech otázky na randomizované hodnoty úloh je báze nepředvídatelná. Není jasné, která proměnná bude v bázi a která se v bázi nenachází. V takovýchto případech lze využít návratové indexy pro navrácení jména bazické proměnné například v prvním řádku. Množství řádků simplexové tabulky se po sestavení výchozí tabulky nikdy nemění. Tato funkcionalita tak řeší problém otázek na případně nerelevantní otázky na nebazické proměnné.

5.6 Další vývoj

Všechny aplikace a pluginy se mohou dále vyvíjet a vylepšovat. V dalších odstavcích jsou popsány neimplementované funkce, které by bylo možné v dalším vývoji přidat. Tyto funkce nebo změny v návrhu by řešily konkrétní problém, který se ukázal v průběhu vývoje nebo testování. Následně je popsáno potenciální přidání rozšíření funkcionality simplexu do oficiální verze otázkového typu formulas.

Prvním nedostatkem odhaleným reálným používáním pluginu, je potenciální potřeba otestovat studentův postup rozličnou otázkou nežli otázkou na hodnotu pivotu. Hodnota pivota ověřuje studentovu znalost postupu simplexového algoritmu, jak získání klíčového sloupce, tak klíčového řádku a kroků potřebných k jejich získání. Pokud plugin ovšem umožní ověření studentova postupu úlohou pouze jedním způsobem, stává se předvídatelná. Úpravou by mohlo být například ukládání všech hodnot jednotlivých simplexových tabulek v průběhu výpočtu. Zároveň by toto řešení poskytovalo větší podporu adaptivního módu, kdy by bylo možné studentovi dávat při chybné odpovědi lepší rady.

Druhým nedostatkem odhaleným v testování je míra požadovaných znalostí o simplexovém algoritmu a jeho zadání. Zadavatel musí umět sestavit úlohu a správně ji převést na kanonický tvar. Podproblémem je časová náročnost. Tento nedostatek lze rozdělit na dvě části. První pro úlohy fixně zadané uživatelem. Pro úlohy fixně zadané zadavatelem je nutné vytvořit proměnné pro každý argument funkce simplex. Pro otázky randomizované je nutné vytvořit proměnné s rozsahem (rozsah generovaných hodnot) pro každý parametr z omezujících podmínek či maximalizační funkce. Útěchou může být tato složitost pouze u první úlohy, neboť je možné nakopírování generování náhodných hodnot ze vzorové úlohy či zadavatelovi jiné úlohy. Jedním z řešení tohoto problému by mohlo být umožnění zadání jednoho argumentu, který by udával množství proměnných (proměn-

né, pomocné, doplňkové). Na základě těchto dat by plugin vygeneroval úlohu. Dále by bylo nutné označit maximalizační či minimalizační úlohu. Takto náhodná úloha s nenáhodnou velikostí by už mohla být vygenerována. S touto funkcionalitou by ovšem musel být umožněn výpis hodnot vygenerovaných proměnných, aby jednotlivé hodnoty mohli být zakomponovány do zadání úlohy a do otázek jednotlivých podúloh.

Bylo by dobré přidat další post optimalizační úlohy. Příkladem by mohlo být o kolik se musí změnit cena proměnné x1, aby vystoupila z báze a jiné. Přidání takovýchto úloh je popsané v kapitole pojmenované přidání dalších indexů. V aktuálním stavu jde tyto post optimalizační úlohy vytvořit také, nicméně si je musí zadavatel složit z hodnot výstupů knihovny. Což v obecné rovině není ideální řešení.

V současném stavu je nadstavba neoficiálně přidána do qtype_formulas. Lze ji volně šířit a používat. Nicméně plugin formulas se vyvíjí dále. Aby tato nadstavba byla zařazena do oficiálního vydání, musela by splňovat následující požadavky. Kód musí splňovat zásady stylu dle oficiálních Moodle stránek. Musel by být vytvořen dokument obsahující teoretické poznatky o simplexovém algoritmu a dokumentaci ke kódu s příklady použití v angličtině.

6 Závěr

Teoretická část práce byla zaměřena na problematiku obecného testování a následně testování znalostí simplexového algoritmu. Další část se zaměřila na LMS Moodle a jeho aktuální funkcionality. Poslední část se zabývala obecným postupem výpočtu simplexového algoritmu a následně aplikováním do reálné úlohy.

Praktická část práce navázala na informace získané z teoretické části, které byly použity na analýzu postupu při testování simplexového algoritmu v LMS Moodle. Jako místo s nejlepším potenciálem pro časovou úsporu byl určen výpočet hodnot simplexového algoritmu. Na základě analýzy bylo navrženo softwarové řešení, popsané v navazujících kapitolách praktické části. Tento návrh softwarového řešení je následně implementován. Software byl následně otestován a byly popsány vzorové úlohy.

Výsledkem práce je plugin redukující časovou náročnost a poskytující obecnou podporu zadávání úloh simplexového algoritmu v LMS Moodle. Plugin zadavateli umožňuje vybrat jednu ze základních návratových hodnot, které může kombinovat například pro složitější simplexové výpočty. Tímto směrem by mohl směřovat budoucí vývoj. Plugin je připravený na jednoduché dodělání dalších postoptimalizačních úloh, aby se dále zefektivnilo zadávání a testování simplexových úloh.

7 Seznam použitých zdrojů

1. Oakeson, I., 2022. Advantages and Disadvantages of a Computer Based PE Exam | Civil Engineering Academy. [online] Civil Engineering Academy. Dostupné z: <https://civilengineeringacademy.com/advantages-and-based-pe-exam/>
2. Wall, J. E. (2000) ,7 Advantages Digital Assessments Have over Paper Tests and Exams | Emerging Education Technologies. Emerging Education Technologies | Engaging Students and Enhancing Learning Outcomes With Instructional Technologies and Active Learning [online]. Copyright © [cit. 23.03.2022]. Dostupné z: https://www.emergingedtech.com/2019/03/7-advantages-digital-assessments-have-over-paper-tests-exams/
3. Paper Based Test or Computer Based Test: Which one is the best? – Prepona News. PREPONA - Certification Platform [online]. Dostupné z: http://www.prepona.com/news/?p=1012&lang=en
4. Mary Burns 15 Benefits Of Computer-Based Testing - eLearning Industry. eLearning Industry - Post your eLearning article. At eLearning Industry you will find the best collection of eLearning articles, eLearning concepts, eLearning software, and eLearning resources. [online]. Copyright © 2011 [cit. 23.03.2022]. Dostupné z: https://elearningindustry.com/15-benefits-of-computer-based-testing
5. Advantages and Disadvateges of computer-based quizzes V.S paper-based | WebheadsinAction.org. WebheadsinAction.org | Communities of Practice Online [online]. Dostupné z: http://www.webheadsinaction.org/content/advantages-and-disadvateges-computer-based-quizzes-vs-paper-based
6. Ben Backes James Cowan, ScienceDirect. ScienceDirect [online]. Copyright © [cit. 23.03.2022]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/S0272775718305119?via%3Dihub
7. Yassin Karay, Stefan K Schauber, Christoph Stosch, Katrin Schüttpelz-Brauns Computer versus paper--does it make any difference in test performance? - PubMed. PubMed [online]. Dostupné z: https://pubmed.ncbi.nlm.nih.gov/25584472/

8. Paper vs. Online Testing: What's the Impact on Test Scores? - FutureEd. FutureEd - A New Voice for American Education [online]. Dostupné z: https://www.future-ed.org/work/paper-vs-online-testing-whats-the-impact-on-test-scores/
9. Docsity.com. 2022. Dual Simplex Method - Systems Analysis - Old Exam Paper - Docsity. [online] Dostupné z: https://www.docsity.com/en/dual-simplex-method-systems-analysis-old-exam-paper/298405/
10. Docsity.com. 2022. Linear Programming - Exam, Exams for Linear Programming [online] Dostupné z: https://www.docsity.com/en/points-linear-programming-exam/254497/
11. About Moodle - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/311/en/About_Moodle
12. Communication Between Components - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Communication_Between_Components
13. Question Engine 2:Design - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Question_Engine_2:Design
14. Question Engine 2:Overview - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Question_Engine_2:Overview
15. Using the Question Bank – Moodle Resource Center. Trinity Banter – A blogging & publishing platform for the Trinity College community. [online]. Dostupné z: https://commons.trincoll.edu/moodle/using-the-question-bank/
16. Docs.moodle.org. 2022. Question bank - MoodleDocs. [online] Dostupné z: <https://docs.moodle.org/311/en/Question_bank>
17. Question Engine 2:How the question engine currently works - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Question_Engine_2:How_the_question_engine_currently_works
18. Overview of the Moodle question engine - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Overview_of_the_Moodle_question_engine

19. Moodle - Quiz activity (staff/faculty) - IT Knowledgebase - RRU IT Services. Home - RRU IT Services [online]. Dostupné z: https://confluence.royalroads.ca/pages/viewpage.action?pageId=8422975
20. Development:Quiz [online]. Dostupné z: https://docs.moodle.org/19/en/Development:Quiz
21. Jenny Spohrer Taking Online Quizzes and Exams in Moodle : Tech Documentation . Tech Documentation [online]. Copyright © 2022 [cit. 24.03.2022]. Dostupné z: https://techdocs.blogs.brynmawr.edu/10398
22. Question engine - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Question_engine
23. Quiz state diagrams - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Quiz_state_diagrams
24. Quiz settings - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/311/en/Quiz_settings
25. Plugin files - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/dev/Plugin_files
26. Question categories - MoodleDocs. 302 Found [online]. Dostupné z: https://docs.moodle.org/29/en/Question_categories
27. Správa úloh - Moodle: návody pro uživatele. Moodle: návody pro uživatele [online]. Dostupné z: https://moodledocs.phil.muni.cz/testove-ulohy/banka-uloh/kategorie-testovych-uloh/
28. Moodle UK pro výuku 1 [online]. Copyright © [cit. 24.03.2022]. Dostupné z: https://dl1.cuni.cz/pluginfile.php/736795/mod_resource/content/0/Novinky%20Moodle%203_6_20190719_revs.pdf
29. Moodle Formulas [online]. Dostupné z: https://moodleformulas.org/course/view.php?id=22§ion=1
30. PHP | Introduction - GeeksforGeeks. GeeksforGeeks | A computer science portal for geeks [online]. Dostupné z: https://www.geeksforgeeks.org/php-introduction/
31. Gaussian elimination - Wikipedia. [online]. Dostupné z: https://en.wikipedia.org/wiki/Gaussian_elimination

32. DML-CZ - Czech Digital Mathematics Library [online]. Copyright © [cit. 24.03.2022]. Dostupné z: https://dml.cz/bitstream/handle/10338.dmlcz/141858/PokrokyMFA_53-2008-3_2.pdf
33. Simplex algorithm - Wikipedia. [online]. Dostupné z: https://en.wikipedia.org/wiki/Simplex_algorithm#History
34. University of Baltimore Home Page web services [online]. Dostupné z: http://home.ubalt.edu/ntsbarsh/opre640a/partviii.htm
35. Friebelová, Jana. 2009. Operační analýza. České Budějovice : JČU Ekonomická fakulta, 2009. ISBN 978-80-7394-193-2.
36. Linear Programming – Explanation & Examples. The Story of Mathematics - A History of Mathematical Thought from Ancient Times to the Modern Day [online]. Copyright © 2020 Luke Mastin [cit. 24.03.2022]. Dostupné z: https://www.storyofmathematics.com/linear-programming
37. Jasbir Singh Arora,2017,Introduction to Optimum Design ISBN 978-0-12-800806-5
38. 2_LP_II.. Mediasite.czu.cz [online]. Dostupné z: https://mediasite.czu.cz/Mediasite/Play/cd4fa4db46ce41e991906b42cc4932661d

8 Seznam obrázků a kódů

8.1 Seznam obrázků

Obrázek 1: Komunikace mezi komponenty [12] 17
Obrázek 2: Povolená komunikace v Moodle [12] 18
Obrázek 3: Stavy a jejich přechody [12] 21
Obrázek 4: Hodnoty jednotlivých stavů [12] 21
Obrázek 5: Quiz stavy [23] 24
Obrázek 6: Pokus stavy [23] 24
Obrázek 7: Kategorie struktura [26] 26
Obrázek 8: Štítky v úloze [vlastní zpracování] 27
Obrázek 9: Štítky v bance úloh [vlastní zpracování] 27
Obrázek 10: Formulas obecná nastavení [vlastní zpracování] 28
Obrázek 11: Formulas proměnné [vlastní zpracování] 29
Obrázek 12: Formulas úvodní text [vlastní zpracování] 29
Obrázek 13: Formulas podotázka [vlastní zpracování] 30
Obrázek 14: Podmínky vzorového příkladu bedýnky (vlastní zpracování) 33
Obrázek 15: Vzorový příklad bedýnky grafické řešení [vlastní zpracování] 34
Obrázek 16:Vzorový příklad bedýnky výpočet reálného řešení [vlastní zpracování] 35
Obrázek 17: Účelová funkce kanonický tvar [37] 35
Obrázek 18: Omezující podmínky kanonická forma [37] 36
Obrázek 19: Podmínky nezápornosti [37] 36
Obrázek 22: Simplexová tabulka prázdná [vlastní zpracování] 39
Obrázek 23: Zlatnictví simplexová tabulka 0 krok. [vlastní zpracování] 40
Obrázek 24: Zlatnictví simplexová tabulka 1 krok. [vlastní zpracování] 41
Obrázek 25: Zlatnictví simplexová tabulka 1 krok přepočet. [vlastní zpracování] 41
Obrázek 26: Zlatnictví simplexová tabulka 2 krok. [vlastní zpracování] 42
Obrázek 27: Zlatnictví simplexová tabulka 3 krok. [vlastní zpracování] 42
Obrázek 28: Zlatnictví výsledná simplexová tabulka. [vlastní zpracování] 42
Obrázek 29: Calculated datasets [vlastní zpracování] 48
Obrázek 30: Návrh syntaxu [vlastní zpracování] 51
Obrázek 31: Simplexová tabulka řádek zj-cj [vlastní zpracování] 53

Obrázek 32: Simplexová tabulka průsečíky [vlastní zpracování] 54
Obrázek 33: Simplexová tabulka hodnota b [vlastní zpracování] 54
Obrázek 34: Simplexová tabulka jméno bazické proměnné [vlastní zpracování] 55
Obrázek 35: Indexy rychlá referenční tabulka [vlastní zpracování] 72
Obrázek 36: Zlatnictví fixní Moodle část 1 [vlastní zpracování] 74
Obrázek 37: Zlatnictví fixní Moodle část 2 [vlastní zpracování] 75
Obrázek 38Zlatnictví fixní Moodle část 3 [vlastní zpracování] 76
Obrázek 39:Zlatnictví fixní Moodle část 4 [vlastní zpracování] 76
Obrázek 40:Zlatnictví fixní Moodle část 5 [vlastní zpracování] 77
Obrázek 41: Zlatnictví náhodné Moodle část 1 [vlastní zpracování] 78
Obrázek 42: Zlatnictví náhodné Moodle část 2 [vlastní zpracování] 79
Obrázek 43: Zlatnictví náhodné Moodle část 3 [vlastní zpracování] 80
Obrázek 44: Zlatnictví náhodné Moodle část 4 [vlastní zpracování] 81
Obrázek 45: Zlatnictví náhodné Moodle část 5 [vlastní zpracování] 82
Obrázek 46: Zlatnictví náhodné Moodle část 6 [vlastní zpracování] 83
Obrázek 47: Zlatnictví náhodné Moodle část 7 správné odpovědi [vlastní zpracování] 83

8.2 Seznam kódu

Kód 1: Kontrola vstupů 1. část 57
Kód 2: Kontrola vstupů 2. část 58
Kód 3: Kontrola vstupů 3. část 58
Kód 4: Proměnné qtype_formulas_simplex_table 59
Kód 5: qtype_formulas_simplex definice proměnných 59
Kód 6: qtype_formulas_simplex inicializace 60
Kód 7: qtype_formulas_simplex unit_matrix 61
Kód 8: : qtype_formulas_simplex possible_index_precalculate 62
Kód 9: qtype_formulas_simplex reverse_zparameters 63
Kód 10: qtype_formulas_simplex create_table 63
Kód 11: qtype_formulas_simplex calculate 64
Kód 12: qtype_formulas_simplex decision část 1 65
Kód 13: qtype_formulas_simplex decision část 2 65
Kód 14: qtype_formulas_simplex decision část 3 66

Kód 15: qtype_formulas_simplex recalculation část 1 67
Kód 16: qtype_formulas_simplex recalculation část 2 67
Kód 17: qtype_formulas_simplex recalculation část 3 68
Kód 18: qtype_formulas_simplex return_answer_index část 1 69
Kód 19: qtype_formulas_simplex return_answer_index část 2 70
Kód 20: qtype_formulas_simplex return_answer_index část 3 70

Přílohy

V přiloženém zip souboru na CD je přiložený plugin. Dále CD obsahuje 1 fixní úlohu a jednu náhodnou úlohu v XML. Obě úlohy jsou popsané v kapitolách příklad pro fixní hodnoty a příklad pro náhodné hodnoty.