Fakulta matematiky, fyziky a informatiky Univerzity Komenského

DOKUMENTÁCIA
ANIMÁCIE MODELOV SOFTVÉROVÝCH ARCHITEKTÚR

Predmet : Tvorba informačných systémov

Autori : Adam Chovanec, Miriam Cidoríková, Marek Ďurana, Fedor Kalantaev

OBSAH

KATALÓG POŽIADAVIEK

1. Úvod

1.1.DÔVODY VZNIKU TOHTO DOKUMENTU

Katalóg požiadaviek vznikol ako záväzný dokument obsahujúci požiadavky a ich jednoznačný opis pred a počas tvorby projektu Animácie modelov softvérových architektúr od zadávateľa vývojárom. Je určený všetkým osobám, ktoré sú zapojené do vývoja tohto projektu a práce s ním.

1.2. ROZSAH VYUŽITA SYSTÉMU

Rozširujeme funkcie systému AnimArch, ktorý dokáže vizualizovať vzťahy medzi jednotlivými triedami v class diagrame. V prvom rade systém slúži ako pomôcka pri výučbe softvérového inžinierstva, presnejšie architektúry softvérových systémov. V druhom rade je systém nástrojom pre výskum fúzie statického a dynamického modelu class diagramu. Môže to byť aj nástroj pri analýze a návrhu architektúry softvérového systému.

1.3.DEFINÍCIE, POJMY A SKRATKY

Class diagram - model, ktorý popisuje štruktúru systému zobrazením tried a vzťahov medzi nimi.

Fúzia - spojenie, prepojenie, spojiť sa

xUML - executable UML; jazyk pre tvorbu vykonateľných modelov softvéru

OAL (Object Action Language) - platformovo nezávislý jazyk kompilovateľný na platformovo závislé jazyky

EA - Enterprise Architect; softvér pre modelovanie softvéru v UML

GUI - Graphical User Interface; používateľské rozhranie pozostávajúce z interaktívnych vizuálnych prvkov

AST - Abstract Syntax Tree; abstraktná reprezentácia výrazov v danom jazyku

API-Application Programming Interface; rozhranie poskytované aplikáciou pre komunikáciu s inými aplikáciami

VS - Visual Studio; vývojové prostredie pre jazyk C#

Debug mód - mód, ktorý zabezpečuje interaktívne zásahy do vykonávania animácie

Formát .oal - formát textového súboru, v ktorom jeden riadok zodpovedá jednému príkazu jazyka OAL.

1.4. ODKAZY

AnimArch - https://github.com/TIS2020-FMFI/uml-3d/tree/master/UnityProjectDP - softvér, ktorý rozširujeme

xtUML - https://xtuml.org/ - výučba executive UML, know how OAL

1.5.OBSAH DOKUMENTU

Druhá kapitola obsahuje všeobecný opis systému rozdelený do nasledovných oblastí:

- všeobecná perspektíva systému
- funkcie systému
- charakteristiku používateľa
- všeobecné obmedzenia systému

V tretej kapitole sa nachádza zoznam požiadaviek, ktoré sú rozdelené do podkapitôl podľa jednotlivých blokov, ktorých sa dané požiadavky týkajú.

2. Všeobecný popis

2.1.PERSPEKTÍVA SYSTÉMU

Aplikácia AnimArch poskytuje samostatnú metódu animácie softvérových architektúr a scenárov. Operuje nad diagramom tried, ktorý animuje na základe príkazov zapísaných v modifikovanom jazyku OAL. Jazyk umožňuje vytvárať jednoduché scenáre (postupnosť animačných krokov), alebo sofistikované scenáre pomocou podmienok, cyklov, paralelizmu, objektov, premenných a výpočtov nad výrazmi. Aplikácia umožňuje pracovať s diagramami vytvorenými v EA priamo pomocou kliknutia v EA.

2.2.FUNKCIE SYSTÉMU

Systém vizualizuje vzťahy medzi jednotlivými triedami v class diagrame. Pracuje s modelmi class diagramov, ktoré boli vytvorené a následne exportované z externých aplikácií (napr. Enterprise Architect). Systém k danému class diagramu vytvorí object diagram v pomere 1:1 tak, že vytvorí inštanciu každej triedy a následne s týmito inštanciami pracuje. Na tvorbu vizualizácie/animácie ponúka systém skript. Animácia je prevedená formou vysvietenia celej triedy a jej metódy. Následne sa postupne vysvieti vzťah medzi triedami. Na záver sa vysvieti druhá trieda a jej metóda. Systém je schopný vysvietiť aj viac tried. Triedy sú vysvietené v poradí, v akom sú v skripte. Skript môže používateľ napísať sám alebo ho systém vygeneruje interaktívnou formou (užívateľ klikne na triedy a metódy, ktoré chce vizualizovať). Všetky príkazy v skripte sú v jazyku OAL a slúžia výhradne na animáciu. Po spustení skriptu prebehne animácia, ktorej dĺžku a výzor (farebná schéma) si volí užívateľ. Animácia prebieha naraz v samotnom class diagrame zvýraznením tried, metód a vzťahov medzi triedami a taktiež aj v skripte zvýraznením jednotlivých príkazov. Systém dokáže skript exportovať do textového súboru(.txt).

2.3.CHARAKTERISTIKY POUŽÍVATEĽA

Systém rozpoznáva 2 roly používateľov:

vývojár animácie - človek, ktorý vytvára skript v jazyku OAL, ktorý animuje class diagram pre používateľa (kolega, študent)

používateľ animácie - človek, ktorý systém využíva na učebné účely/analýzu už existujúceho class diagramu

2.4.VŠEOBECNÉ OBMEDZENIA

Jediné obmedzenie, ktoré stanovil zadávateľ je, že systém musí bežať na operačnom systéme, ktorý podporuje C#, Unity, Enterprise Architect (alebo ekvivalentný softvér na tvorbu diagramov) a funkčnosť vo virtuálnej realite.

3. Požiadavky

Požiadavky sú rozdelené na konkrétne požiadavky pre vývojára a pre používateľa, keďže obe tieto roly majú v systéme rozličné funkcie a využívajú rozličnú funkcionalitu systému.

3.1. IMPLEMENTÁCIA DEBUG MÓDU PRE VÝVOJÁRA

3.1.1. V aplikácii bude pri prehrávaní animácie umiestnené okno so skriptom v používateľskom rozhraní (bližšie v návrhu používateľského rozhrania).

3.1.2. Aplikácia bude vysvecovať tie časti (príkazy) skriptu, ktoré sú práve zobrazované animáciou v diagrame (animácia je bližšie popísaná v časti **2.2**). Vysvietený bude celý príkaz alebo jednotlivé časti príkazu (realizácia je daná na vývojároch).

3.2. IMPLEMENTÁCII DEBUG MÓDU PRE POUŽÍVATEĽA

3.2.1. Aplikácia bude umožňovať vývojárovi animácie zadefinovať metódy do skriptu OAL. Metódy sa budú skladať z hlavičky a tela.

3.2.2. Aplikácia vytvorí debug okno stavov, kde sa bude zobrazovať názov aktuálne animovanej triedy, obsah jej jednotlivých atribútov a názov spúšťanej metódy.

3.2.3. Aplikácia zobrazí inštanciu triedy v používateľskom rozhraní ako podtabuľku atribútov a ich hodnôt pre danú triedu.

3.2.4. V aplikácii sa bude nachádzať tlačidlo s názvom “Create method body”. Po jeho zvolení je možné kliknúť na triedu a jednu jej metódu. Následne sa okno s OAL skriptom prekryje rovnakým oknom, ktorého obsah bude prázdny. Nad oknom bude názov vybranej triedy a jej metódy. Toto okno je možné zatvoriť, čím sa znova otvorí okno s OAL skriptom.

3.2.5. Tlačidlo “Create method body” bude umiestnené v riadku s tlačidlami Check a Save pod oknom pre OAL skript. Vzhľad bude mať rovnaký ako tieto dve pôvodné tlačidlá.

3.2.6. V OAL skripte bude možné zadefinovať telo metódy. V tele danej metódy môžeme definovať for cykly, if podmienky a aj lokálne premenné.

3.2.7. Pri volaní metódy v skripte môžeme znakom (!) označiť, že chceme animovať aj telo danej metódy. Ak telo metódy nechceme animovať, za volaním metódy bude len prázdna zátvorka ().

3.2.8. Používateľ má možnosť skript metódy uložiť. Skript sa uloží ako súbor vo formáte “MenoTriedyMenoMetody.oal” na používateľom zvolené miesto na disku.

3.2.9. Počas animácie metódy sa otvorí okno s jej skriptom, v ktorom sa budú postupne vysvecovať vykonávané príkazy.

3.2.10. Vizualizácia atribútov metód a ich prípadnych zmien, pri vyvolávaní daných metód. Vizualizácia bude prebiehať zobrazením tabuľky atribútov metódy a ich aktuálnych hodnôt v používateľskom rozhraní. Pri zmene hodnoty atribútu sa hodnota v tabuľke aktualizuje.

3.2.11. Atribúty metód bude možné v tabuľke editovať.

4. Prílohy

https://github.com/TIS2020-FMFI/uml-3d/blob/master/prilohy/OAL_manual.pdf - OAL manual

https://github.com/TIS2020-FMFI/uml-3d/blob/master/prilohy/dokumentacia_prototypu.pdf - dokumentácia prototypu

NÁVRH

1. Špecifikácia vonkajších interfejsov

1.1. IMPORT/EXPORT SKRIPTU PRE ANIMÁCIU

Aplikácia komunikuje s úložiskom na zariadení a je schopná načítať/uložiť skript z/do súboru typu .oal (textový súbor, v ktorom 1 riadok je 1 príkaz jazyku OAL), ktorý je možné vybrať si v okne so súbormi. Toto okno je realizované použitím externého assetu File Browser, ktorý ponúka prostredie Unity.

1.2. IMPORT CLASS DIAGRAMU

Aplikácia pracuje s diagramami, ktoré sú uložené v súboroch typu .xml. Tieto súbory sú do aplikácie importované z externého programu na tvorbu diagramov (napr. EA).

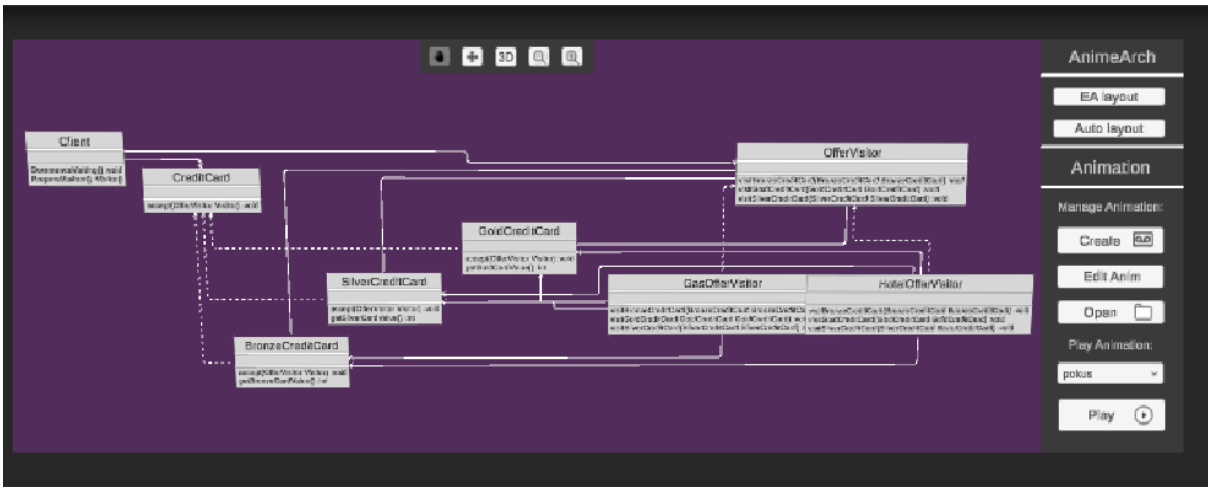
1.3. KOMUNIKÁCIA S PERIFÉRIAMI POČÍTAČA

Systém taktiež komunikuje s myšou pri tvorbe animácie interaktívnou formou (používateľ vykliká triedy a metódy, ktoré chce animovať) a s klávesnicou pri tvorbe OAL skriptu pre animáciu.

2. Návrh používateľského rozhrania

Používateľ si najskôr musí vytvoriť diagram tried. Túto funkcionalitu neposkytuje AnimArch. Využívame už existujúci editor v EA. Používateľ si teda vytvára nový diagram v EA, alebo využije svoj už existujúci diagram. Prostredníctvom GUI EA zvolí možnosť pre spustenie AnimArchu, ktorý následne zobrazí daný diagram v Unity okne, ktoré sa otvára ako súčasť GUI EA. Dáta o diagrame putujú do AnimArchu vo forme XMI exportu poskytovaného EA API.

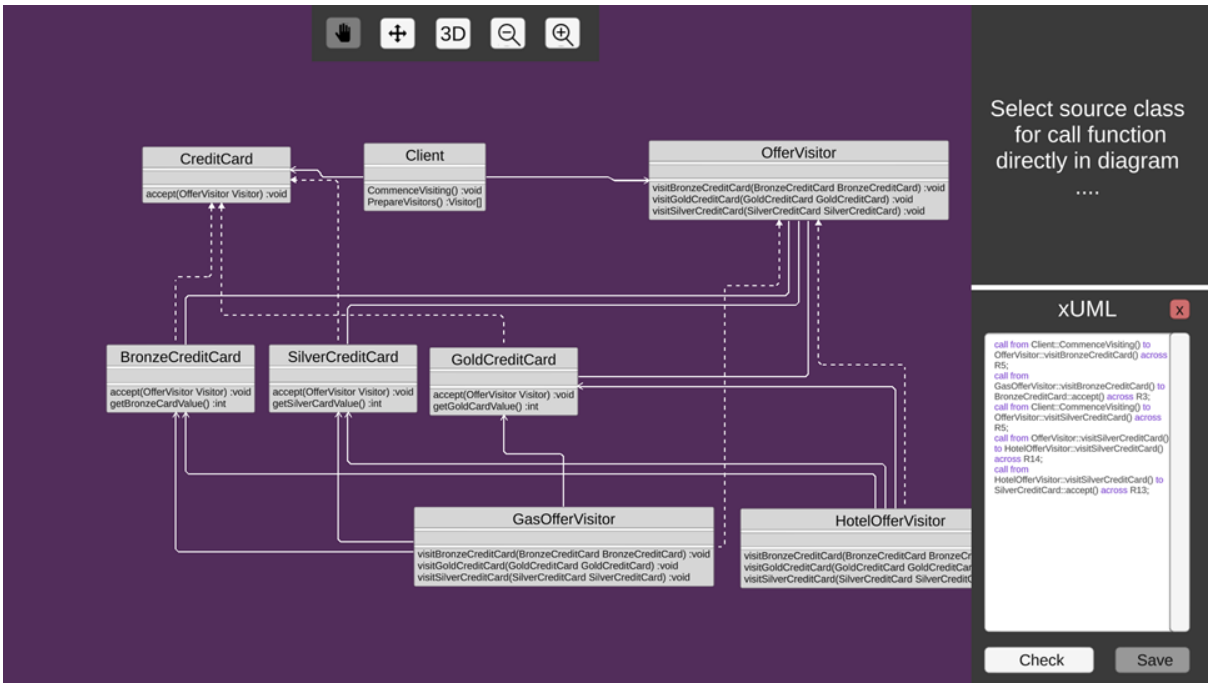
2.1. AKTUÁLNY VZHĽAD



Obr.1 : Obrazovka po otvorení diagramu v AnimArch-u

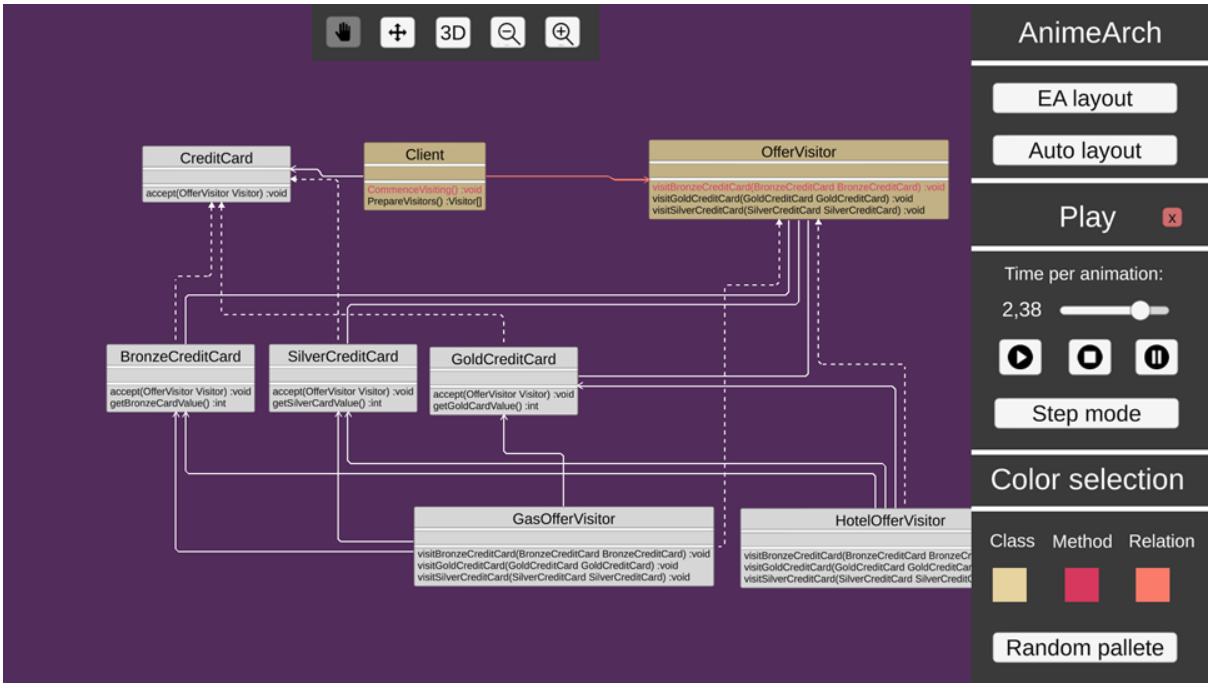
Uprostred sa nachádza diagram používateľa, na pravej strane sa nachádza panel s nástrojmi. Diagram obsahuje vždy obdĺžnik, ktorý reprezentuje triedu, ktorej názov je v nej napísaný a pod názvom sa nachádzajú názvy metód, ktoré môže vykonať.

Našou úlohou je do takéhoto prostredia doimplementovať vysvecovanie riadkov scriptu, ktorý napíše používateľ.



Obr.2 : Obrazovka po kliknutí na Create na paneli nástrojov.

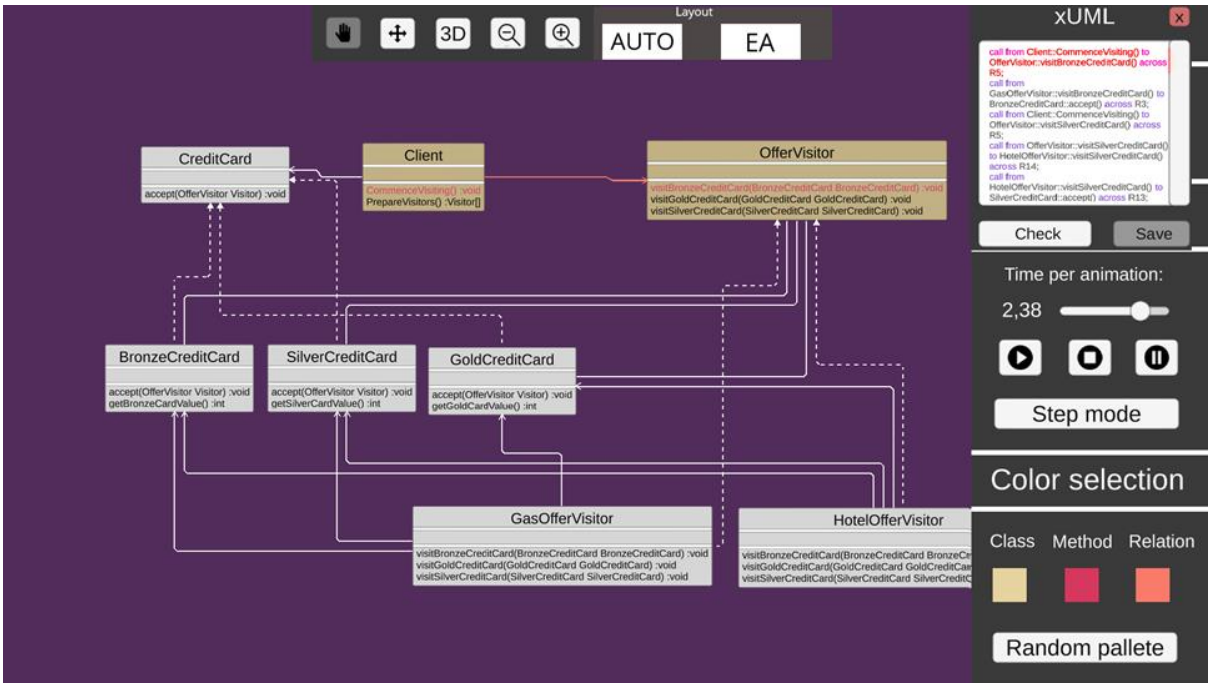
Vidíme priestor pre napísanie scriptu, ktorý budeme vysvecovať.



Obr.3 : Obrazovka po kliknutí Play, priebeh animácie

Do takto vyzerajúcej obrazovky sa budeme snažiť umiestniť okno so scriptom, v ktorom sa bude vysvecovať práve vykonávaná časť.

2.2. NÁVRH NA UMIESTNENIE OKNA NA VÝPIS SCRIPTU A JEHO VYSVECOVANIE



Obr.4 Návrh umiestnenia okna s výpisom scriptu

Presunúť možnosti rozloženia na hornú lištu a na ich pôvodné miesto umiestniť výpis OAL kódu.

Kód sa bude vysvecovať tak zmenou farby písma.

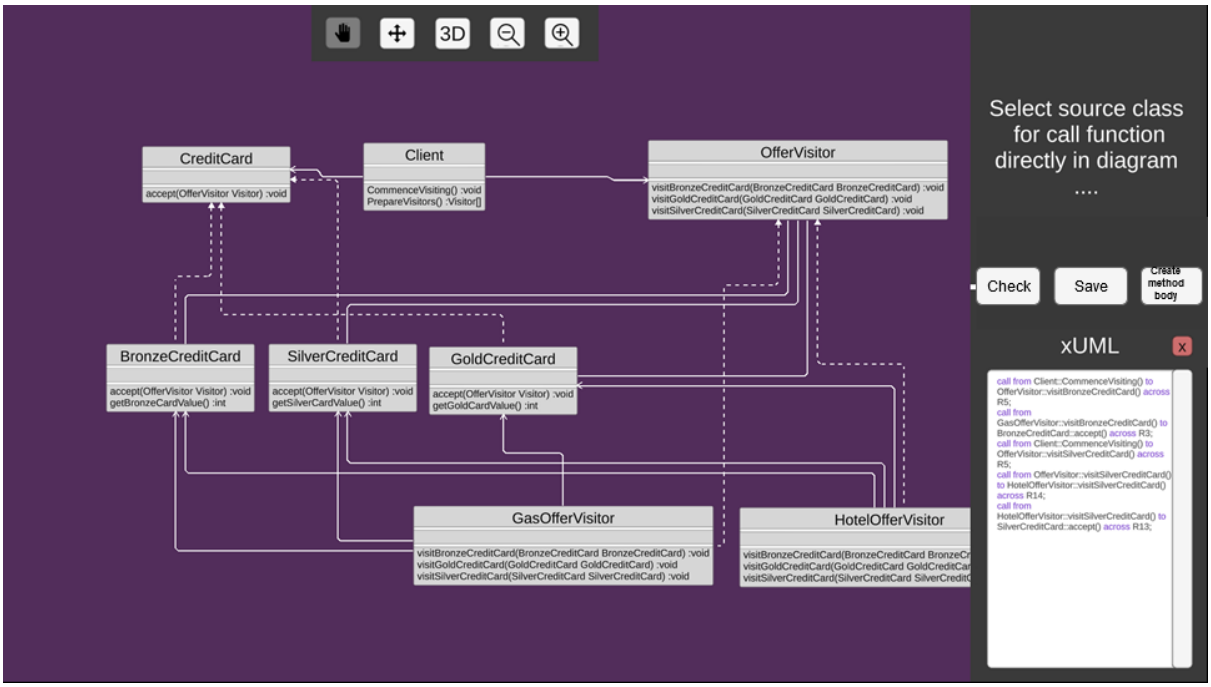
Obr. 5 : Zmena farby písma

2.3. NÁVRH GRAFICKÉHO ROZHRANIA DRUHEJ POŽIADAVKY

Pridáme tlačidlo – „Create method body“. Po jeho zvolení môžeme kliknúť na triedu – rovnaká voľba metódy ako pri tvorbe animácie, ale volíme len jednu triedu a jednu metódu, nie dvojicu.

Po zvolení prekryjeme okienko s OAL skriptom, rovnakým okienkom, ale jeho obsah bude prázdny (príp. naplnený už existujúcim OAL skriptom pre danú metódu). Nad ním musí byť nejaký label s názvom triedy a metódy. Používateľ má možnosť tento skript uložiť a zatvoriť okienko (čo mu opäť ukáže celkový OAL skript).

Toto isté treba spraviť pri vykonávaní tela metódy, pri vysvecovaní riadkov skriptu ako v úlohe 1.


3D
CreditCard
accept(OfferVisitor Visitor) :void
Client
CommenceVisiting() :void
PrepareVisitors() :Visitor[]
OfferVisitor
visitBronzeCreditCard(BronzeCreditCard BronzeCreditCard) :void
visitGoldCreditCard(GoldCreditCard GoldCreditCard) :void
visitSilverCreditCard(SilverCreditCard SilverCreditCard) :void
BronzeCreditCard
accept(OfferVisitor Visitor) :void
getBronzeCardValue() :int
SilverCreditCard
accept(OfferVisitor Visitor) :void
getSilverCardValue() :int
GoldCreditCard
accept(OfferVisitor Visitor) :void
getGoldCardValue() :int
GasOfferVisitor
visitBronzeCreditCard(BronzeCreditCard BronzeCreditCard) :void
visitGoldCreditCard(GoldCreditCard GoldCreditCard) :void
visitSilverCreditCard(SilverCreditCard SilverCreditCard) :void
HotelOfferVisitor
Select source class for call function directly in diagram ....
Check
Save
Create method body
xUML
call from Client::CommenceVisiting() to OfferVisitor::visitBronzeCreditCard() across R5;
call from GasOfferVisitor::visitBronzeCreditCard() to BronzeCreditCard::accept() across R3;
call from Client::CommenceVisiting() to OfferVisitor::visitSilverCreditCard() across R5;
call from OfferVisitor::visitSilverCreditCard() to HotelOfferVisitor::visitSilverCreditCard() across R14;
call from HotelOfferVisitor::visitSilverCreditCard() to SilverCreditCard::accept() across R13;

3. Návrh implementácie

3.1. VYSVECOVANIE RIADKOV V OAL

3.1.1. Návrh prvý

- presunuť tlačidlá EA layout a Auto layout
- na miesto tých tlačidiel umiestnit textový panel

3.1.2. Návrh druhý

- v ľavom hornom rohu
- môžeme pridať malú šípku, aby používateľ mohol toto okno zobraziť alebo skryť

3.2. VYSVECOVANIE RIADKOV KÓDU V TEXTOVOM PANELI.

3.2.1. Návrh prvý. Rekonštrukcia skriptu

- Ked spúšťame animáciu v pamäti máme reprezentaciu skriptu vo forme vlastných tried (*EXEScope*, *EXECommand* - podtypy)
- Každej triede implementujeme metodu *toOALCode()* -> vie zrekonštruovať samú seba
- Zrekonštruovaný skript si uložíme ako pole stringov
- Už takú triedu máme v *EXECommandCall.cs* vola sa *String ToCodeSimple()*
- Každá trieda *EXECommand* si zapamätá index do toho pola
    - Pripravíme si pole stringov v mieste , kde vyvolávame rekonštrukciu kódu...a kde to vlastne bude? Tam, kde spúšťame animáciu po kliknutí na “Play”. Hneď aj vyvoláme konverziu na stringy.
    - **Assets/Scripts/Animation/Animation.cs** -metoda *public IEnumerator Animate()*
        - List<String> OALScript = new List<String>();
        - IntegerWrapper Counter = new IntegerWraper(0);
    - OALProgram.Instance.SuperScope.toArrayCode(OALScript, Counter);
    - Vytvoríme jednoduchú triedu IntegerWrapper(ako trieda Integer v Jave)

- Pridáme metódu *toArrayCode(List<String>OALScript, IntegerWrapper Counter)* triedam *EXECommand, EXEScope*. Pridáme im aj atribút typu *List<int>*,nazveme ho napr. *ScripIndexes*
    - *ScriptIndexes* táto trieda si bude pamätať na ktorom indexe v poli sa nachádza
- Prečo pole indexov a nie len 1 index ? Lebo *EXEScope* maju otvárací aj uzatvárací riadok (if,for,while)

- Metoda toArayCode() pre EXECommand

```
virtual toArrayCode(List<String> OALScript, IntegerWrapper Counter){

    OALScript.append(this.ToCodeSimple());

    Counter.increment();

    this.ScriptIndexes.append(Counter.value());

}
```

- Metóda *toArrayCode()* pre *EXEScope.* Treba pridať abstraktné metódy *openingSyntax()* a *closingSyntax().* Podtypy si ich implementujú, aby vrátili stringy, napr. pre EXEScopeLoopWhile budú vracať “while” + *this.Condition.toCode()* + “;” a “end while;”
    - *openingSyntax()* bude vracať napr. “while” + *this.Condition.toCode()* + “;”
    - *closingSyntax()* bude vracať napr. “end while;”

```
toArrayCode(List<String> OALScript, IntegerWrapper Counter){

    OALScript.append(this.openingSyntax());

    Counter.increment();

    this.ScriptIndexes.append(Counter.value());

    foreach (EXECommand Command in this.Commands){

        Command.toArrayCode(OALScript, Counter);
```

```
            }

            OALScript.append(this.closingSyntax());

            Counter.increment();

            this.ScriptIndexes.append(Counter.value());

        }
```

- Keď je vykonávaná , vytiahneme si z nej tento index a ten vysvietime

- Po jej vykonaní, zhasneme tento index a vykonávame ďalšiu triedu
- Majme metódu napr. *FillTextFields(List<int> CurrentIndexes, List<String> Script)*, ktorá urobí z poľa stringov jeden string (OAL skript), a naformátuje ho tak, aby boli vysvietené riadky označené v *CurrentIndexes* . Potom tento naformátovaný text zobrazí do okienka s textom (text prepíše, nepridáva ho).
- **Assets/Scripts/Animation/Animation.cs** v metóde *public IEnumeratorAnimate()* je cyklus volania príkazov wrappnutych v *AnimationCommand.*

```
ExecuteCommand(AnimationCommand Command, List<String> Script){

        Command.Execute();

        FillTextField(Command.DecoratedCommand.ScriptIndexes, Script);

}
```

- **Assets/Scripts/Animation/Animation.cs -** tu nahradíme *Command.Execute()* našou novou metódou *ExecuteCommand(Command, Script)* na oboch miestach
- to je :

```
if(AnimationSequence[0].isCall)

{
```

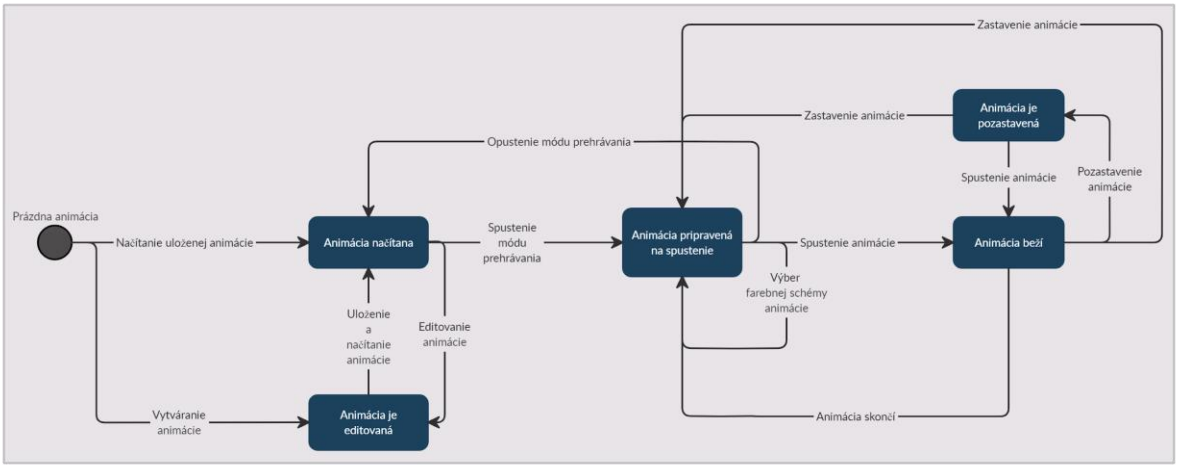
```
        foreach(AnimationCommand Command in AnimationSequence)
        {
            StartCoroutine(Command.Execute());
        }
        yield return StartCorotine(BarierFillCheck());

}
else
{
        foreach(AnimationCommand Command in AnimationSequence)
        {
            Command.Execute();
        }
}
```

- Pred začiatkom cyklu budeme mať pripravené pole stringov

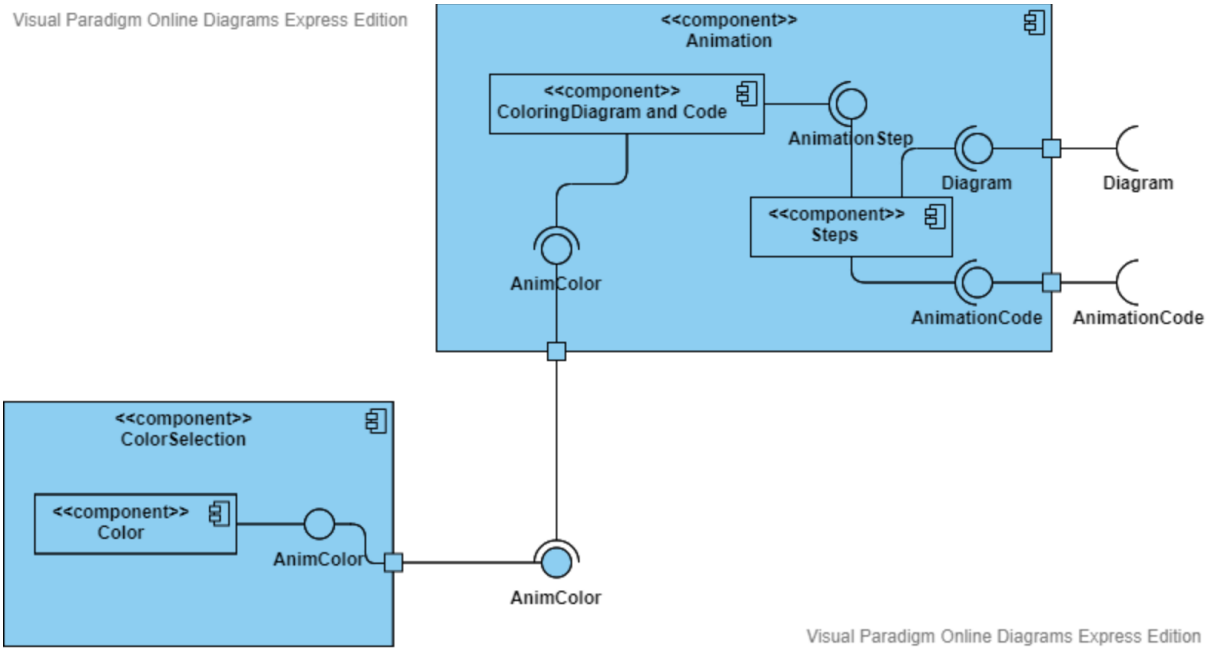
    - Iné príkazy ako vysvietenie sa udejú okamžite, preto len bliknú. Možno by sme mali pridať krátky wait pri iných typoch príkazov ako vysvietenie, aby sa aj tie trošku vysvietili ?

3.3. DIAGRAMY

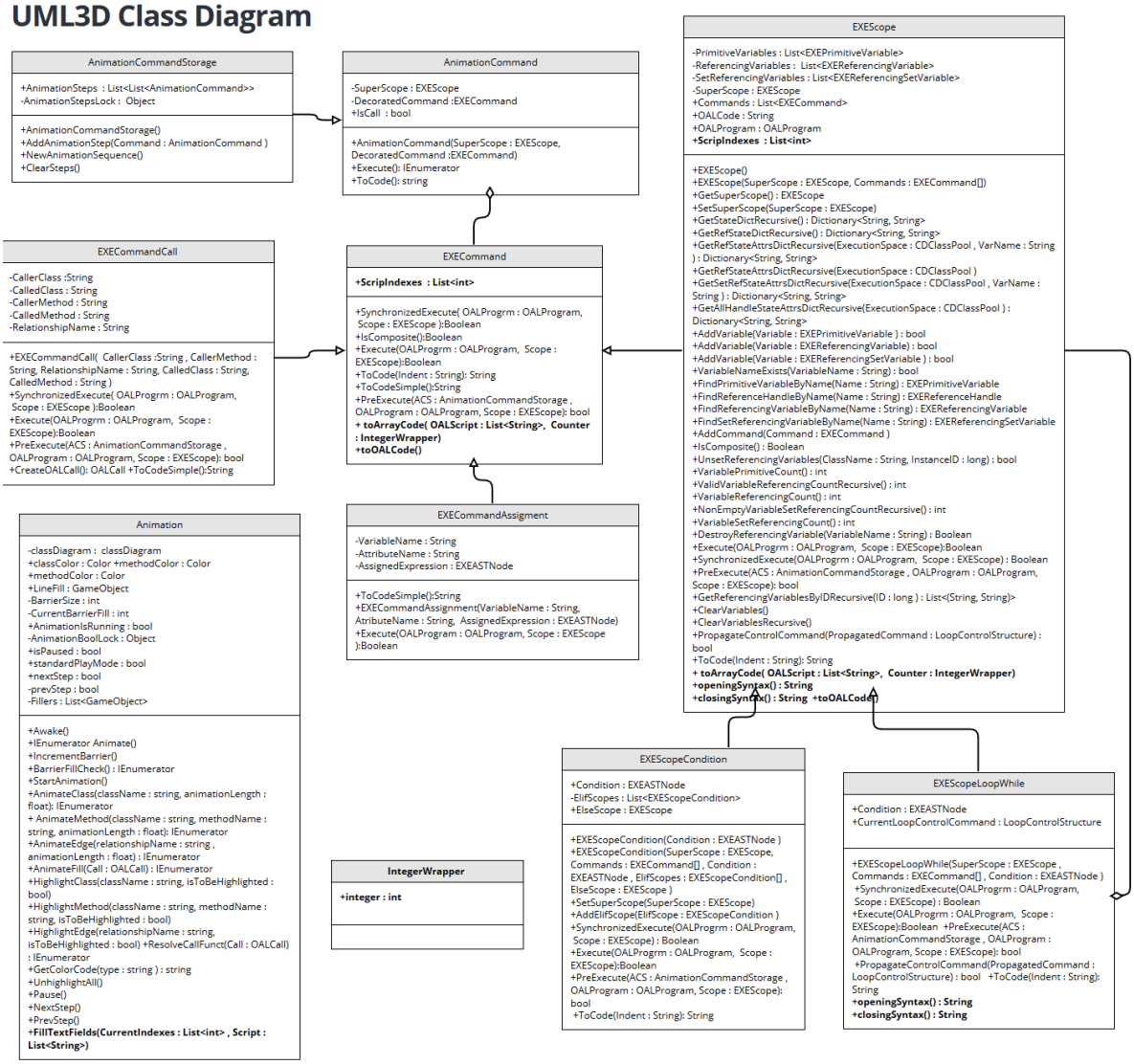
3.3.1. Stavový diagram



3.3.2. Komponentný diagram



3.3.3. Triedny diagram



TESTOVACIE SCENÁRE

1. Debug mód pre vývojára

1.1. TEST SPUSTENIA ANIMÁCIE

a) používateľ spustí animáciu napísaním OAL kódu alebo jeho vygenerovaním
b) OAL kód následne uloží v ľubovoľne vybranom adresári
c) vloží vytvorený súbor pomocou tlačidla open
d) kliknutím tlačidla PLAY sa animácia začne vykonávať
e) butony s možnosťami rozloženia sa presunú na hornú lištu
f) na ich mieste je okno s OAL kódom

1.2. TEST VYTVORENIA SKRIPTU

a) skript sa dá vytvoriť pomocou tlačidla CREATE
b) OAL kód užívateľ napíše do prázdneho modulu XUML alebo vygeneruje vyklikaním z ponuky pomocou modulu search
c) použitím tlačidla check sa skontroluje kód

1.3. TEST ULOŽENIA SKRIPTU

> 2. c.

a) po vytvoreni skriptu sa súbor (.txt) uloží do vybraného adresára, ktorý si vyberá používateľ (tlačidlo SAVE)
b) súbor zostáva uložený, pokiaľ ho niekto fyzicky nezmaže

1.4. TEST BEHU ANIMÁCIE

> 1. d.

a) v okne s napísaným/vygenerovaným skriptom (XUML)sa časovo synchronizovane s animáciou vyfarbuje OAL kód podľa návrhu používateľského rozhrania
    i. každý jeden príkaz (call) je zvýraznený práve vtedy, keď je jeho reprezentácia zvýrazňovaná na diagrame
    ii. po skončení vysvietenia príkazu na diagrame sa príkaz v skripte (XML) vráti do pôvodného stavu

b) po skončení animácie OAL kód ostane v pôvodnom stave

1.5. TEST KOREKTNÉHO NAČÍTANIA SÚBORU

> 1. c

a) užívateľ vyberie súbor pomocou tlačidla OPEN, pokiaľ žiaden neexistuje - (>2. a. - 3.b.<)
b) otvorí sa vyhľadávací modul Unity, kde korektne uložený súbor (> 2. a. - 3.b.<) užívateľ nájde
c) vznikne nový modul (XUML) s textom s načítaného súboru
d) text v XML module nie je rozdielny od uloženého textu v textovom súbore
e) text je rovnako formátovaný ako text v uloženom textovom súbore

1.6. TEST PREKRÝVANIA

>4. a.

a) vzniknutý modul XUML neprekrýva žiaden už existujúci modul rozhrania
b) presunutý modul s možnosťami rozloženia neprekrýva žiaden existujúci modul rozhrania
c) žiaden modul rozhrania neprekrýva diagram v používateľskom rozhraní

1.7. TEST ČITATEĽNOSTI

>1. d.

a) vysvecovaný riadok v OAL skripte je vysvecovaný odlišnou farbou ako pôvodne vypísaný riadok
b) po skončení vysvietenia riadku sa farba riadku opäť zmení na pôvodnú

1.8. TEST KOREKTNÉHO SKONČENIA ANIMÁCIE

>4. b.

a) po kliknutí na krížik tlačidla PLAY s rozhrania zmizne modul (XML) s OAL kódom
b) modul s možnosťami rozloženia sa vráti na pôvodné miesto z pred spustenia animácie (< 1. d.)

1.9. TEST KOREKTNÉHO OAL SKRIPTU

>4. a..

a) skript spĺňa syntax jazyka OAL
b) každý príkaz skriptu musí spĺňať formát jazyka OAL
c) názvy metód a tried musia byť obsiahnuté v načítanom diagrame

vzor:

call from Client::PrepareVisitors() to CreditCard::accept() across R6;
call from GoldCreditCard::getGoldCardValue() to
OfferVisitor::visitGoldCreditCard() across R11;
call from GasOfferVisitor::visitSilverCreditCard() to
BronzeCreditCard::getBronzeCardValue() across R3;

1.10. TEST NESPRÁVNEHO OAL SKRIPTU

a) po zásahu do korektne uloženého OAL skriptu a následnom spustení animácie animácia spadne na chybe

vzor:

“**coment**”
“**random text**” call from Client::PrepareVisitors() to
CreditCard::accept() across R6;
2.
call from GoldCreditCard::getGoldCardValue() to
OfferVisitor::**nonexistingmethod**() across R11;
3.
neprimerane veľký počet volaní

2. Debug mód pre používateľa Implementácia tiel metód

2.1. TEST FORMÁTU TELA METÓD

a. Ak metóda má zadefinované telo
    i. jej telo je zobrazené v samostatnom module
    ii. label modulu je názvom danej metódy
b. Ak metóda nemá zadefinované telo
    i. telo modulu je prázdne
    ii. label modulu je názvom danej metódy

2.2. TEST PRIDANIA TELA METÓDY

a) kliknutím na metódu v diagrame sa otvorí plávajúci modul
b) ak metóda nemá uložené telo, otvorí sa prázdne editovacie okno (modul)
c) ak už existuje telo metódy, otvorí sa editovacie okno s jeho obsahom
d) v tele modulu môžeme editovať resp. pridať telo metódy

2.3. TEST ULOŽENIA TELA METÓDY

-2.d.

a) ak je dokončené editovanie, použijeme tlačidlo SAVE
b) ak je telo pridávané prvýkrát, telo metódy sa uloží do vybraného súboru (.oal), v adresári projektu
c) ak je telo upravované, uloží sa do pôvodného, už vytvoreného súboru s upravenými zmenami

2.4. TEST BEHU ANIMÁCIE

a) v okne s napísaným/vygenerovaným skriptom (XUML)sa časovo synchronizovane s animáciou vyfarbuje OAL kód podľa návrhu používateľského rozhrania
b) v module pribudne rozbaľovacia ponuka s všetkými metódami vyskytujúcimi sa v animácii
c) po kliknutí na niektorú z metód sa otvorí nové plávajúce okno
    a. ak metóda má definované telo, zobrazí sa v plávajúcom module, label modulu je názov metódy
    b. ak metóda telo definované nemá, zobrazí sa modul s lavel názvom metódy a prázdnym telom

d) po skončení animácie OAL kód ostane v pôvodnom stave, otvorené metódy zostanú otvorené

2.5. TEST FORMÁTU TELA METÓDY

a) telo metódy spĺňa syntax jazyka OAL

b) každé volanie v tele metódy musí smerovať z danej metódy

2.6. TEST KOREKTNÉHO TELA METÓDY

a) Názov/label: `Game::CreateArmy()`

b) obsah:

```
call from Game::CreateArmy() to
AbstractFactory::CreateWarrior() across R4;
call from Game::CreateArmy() to
AbstractFactory::CreateWarrior() across R4;
call from Game::CreateArmy() to
AbstractFactory::CreateWarrior() across R4;
call from Game::CreateArmy() to
AbstractFactory::CreateRanger() across R4;
call from Game::CreateArmy() to
AbstractFactory::CreateRanger() across R4;
call from Game::CreateArmy() to
AbstractFactory::CreateMage() across R4;
```

2.7. TEST NESPRÁVNEHO FORMÁTU TELA METÓDY

a) názov/label:

```
AbstractFactory::CreateRanger()
```

b) telo:

```
call from Game::CreateMage() to TrollFactory::CreateRanger()
across R2;
```