Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií
Ilkovičova 2
842 16 Bratislava 4

Tímový projekt 2016-2017:

Tím 13: EduSim

VEDÚCI:
ING. MAREK LÓDERER

ŠTUDENTI:
ADAM BLAŠKO
TOMÁŠ LIŠČÁK
BRANISLAV MAKAN
MARTIN CVIČELA
IVAN GULIS
MAREK MATULA

KONTAKTNÝ EMAIL: MAMBIT@GOOGLEGROUPS.COM

Obsah

1 Dokumentácia k riadeniu

1.1 Úvod k riadeniu projektu

Nikto z nás nie je dokonalý, každý má svoje chyby. Na druhej strane sú veci, v ktorých daný človek vyniká a ktoré mu idú lepšie. Toto platí v každom odvetví a často sa to ukáže, keď ľudia pracujú v tíme. V oblasti IT to platí o to viac. Nato, aby sme čo najlepšie využili našu silu, musíme spolupracovať a navzájom sa riadiť. Tento dokument opisuje riadenie v našom tíme.

Každý člen týmu má okrem iného svoju špecifickú rolu. To neznamená, že na danej veci pracuje len on, ale že má za ňu zodpovednosť. V tímovej práci je veľmi dôležité navzájom sa dopĺňať a určiť si, kto za čo zodpovedá. Role sú špecifikované v časti Role členov tímu a podiel práce.

Ďalej je dôležité zdielať svoj progres, súbory, nápady, myšlienky či problémy. Nato nám slúžia manažovacie nástroje, ktoré sú opísané v časti Aplikácie manažmentov.

Náš tím vyvíja agilným spôsobom. Využívame SCRUM metodiku, pravidelne sa stretávame každý týždeň a úlohy si plánujeme na dvoj-týždňové šprinty. Šprinty sú bližšie opísané v časti Sumarizácie šprintov.

Aby všetko fungovalo správne, potrebujeme dodržiavať konzistentnosť a dodržiavať nejaké pravidlá. Na to sme vytvorili metodiky, ktoré každý člen tímu musí dodržiavať. Dodržiavanie metodík je veľmi dôležitá vec na zabránenie vzniku zbytočných konfliktov v tíme, prípadne zabránenie zbytočnej práci na tej istej veci viac krát. Metodiky sú opísané v časti Používané metodiky - referencie na dokumenty s metodikami.

Na koniec šprintu po spojení našich individuálnych prác do jedného celku robíme retrospektívu. Retrospektíva sa robí po každom šprinte aj globálne za celé fungovanie tímovej práce. V nej sumarizujeme to, čo sme robili dobre, to, v čom sme sa zlepšili a to čo ešte musíme do budúcnosti zlepšiť. Toto je opísané v časti Globálna retrospektíva ZS/LS.

Časť Príloha: Preberacie protokoly obsahuje protokoly o prebratí funkčných verzií zákazníkom, ktorým je firma ATOS.

1.2 Role členov tímu a podiel práce

Meno	Rola
Adam Blaško	Manažér source-control a JIRA
Martin Cvičela	Manažér testovania
Ivan Gulis	Manažér plánovania
Tomáš Liščák	Manažér dokumentácie

Branislav Makan	Manažér webu
Marek Matula	Manažér code-review a komunikácie
Časť dokumentácie	**Vypracoval**
Úvod k riadeniu projektu	Martin Cvíčela
Role členov tímu a podiel práce	Ivan Gulis
Aplikácie manažmentov	Tomáš Liščák
Sumarizácie šprintov	Marek Matula
Používané metodiky - referencie na dokumenty s metodikami	Ivan Gulis
Globálna retrospektíva ZS/LS	Adam Blaško
Úvod k inžinierskemu dielu	Branislav Makan
Globálne ciele pre ZS/LS	Adam Blaško
Celkový pohľad na systém	Ivan Gulis
GUI - grafická časť	Ivan Gulis
GUI - logická časť I	Ivan Gulis
Lokalizácia - resource files	Adam Blaško
Properties Window	Branislav Makan
Pracovná plocha	Branislav Makan
Vykreslovanie čiar	Martin Cvičela
Back-End simulácie elektrického obvodu	Marek Matula
Grafické prvky súčiastok - napojenie na logiku	Marek Matula, Tomáš Liščák

Simulácia el. obvodu v grafickom prostredí	Tomáš Liščák
Súčiastky	Tomáš Liščák
Kompletizácia grafiky I	Ivan Gulis
Skratky - hotkeys	Adam Blaško
Meracie zariadenia	Marek Matula

1.3 Aplikácie manažmentov

1.3.1 Inicializácia a využívanie prostredí

V inicializačnej fáze projektu a procesu vývoja produktu sa nastavili prostredia, ktorých nasadenie mal na starosti Adam Blaško. Prostredia zabezpečujú kolaboratívnu prácu na projekte a šprintoch, dokumentácii, metodikách, code review a samotných comitov zdrojového kódu vytváraného projektu. Jednotlivé prostredia sú:

JIRA - Prostredie pre manažment šprintov(tvorba backlogu a používateľských príbehov, exporty úloh a diagramy postupu vývoja). Toto prostredie bolo najskôr nasadené na školský server a boli v ňom vytvorené kontá pre každého člena tímu. Jeho použitie bolo konzistentné počas všetkých šprintov a postup práce bol vždy:

1. Doplnenie stories do product backlogu
2. Ohodnotenie stories a pridanie taskov k nim spolu s popismi
3. Editácia vlastností stories, priradenie členovy a odhad časovej náročnosti a s tým súvisiace logovanie času práce
4. Zakončenie šprintu a generovanie burndown chartu

Confluence - Prostredie na prácu s dokumentami ako sú metodiky a samotná dokumentácia. Metodiky riadenia tímu boli vytvorené v rámci prvých dvoch šprintov a následne sa nimi tým riadil pri napr. Code review, samotnom písaní kódu a vykonávaní úloh šprintov.

Fish eye + Crucible - Prostredie na správu code review. Začalo sa využívať počnúc druhým šprintom a postup práce je:

1. Vytvorenie review určitej vetvy, alebo comitu
2. Priradenie reviewerov
3. Vykonanie review reviewermi
4. Vyriešenie pripomienok a komentárov v kóde členmi zodpovednými za danú vetvu, alebo komit
5. Označenie komentárov ako resolved a označenie review reviewermi za resolved

Bitbucket - Prostredie na správu git repozitára. Bolo nevyhnutne využívané od začiatku prvého šprintu, nakoľko sa už vtedy začal vytvárať kód. Metóda vývoja je rozdelenie vetiev na jednotlivé

feature vetvy prác na projekte, zatiaľ čo každej feature vetve väčšinou vždy zodpovedal jeden modul produktu. Po skončení prác na feature vetve sa táto merge-uje do develop vetvy a takto sa na-merge-ujú všetky feature vetvy na konci každého šprintu, pokiaľ je daná práca a teda aj úloha šprintu na tejto vetve úspešne ukončená. Po istých míľnikoch produktu sa stav v develop vetve môže merge-núť do master vetvy, ale zatiaľ sa tak ešte neudialo.

1.3.2 Vytváranie zápiskov

V priebehu každého tímového stretnutia sa zapisuje jeho priebeh do formy neštrukturovaných zápiskov, ktoré sa po stretnutí zformalizujú do šablóny a vyvesia na webový server. Zápisky robí akákoľvek osoba chce, formalizáciu robí Tomáš Liščák a vyvesenie Branislav Makan.

1.3.3 Komunikácia

Tímová komunikácia prebieha od prvého týždňa mimo stretnutí pomocou programu hipchat, v ktorom bolo založených niekoľko chatových miestností, každá s odlišnou témou na komunikáciu. Do týchto miestností bol okrem členov tímu povolený vstup aj vedúcemu týmu.

1.4 Sumarizácie šprintov

1.4.1 Šprint Audi

Šprint Audi bol úplne prvý šprint ktorý sa začal 10.10.2016 a bol úspešne ukončený na stretnutí so spoločnosťou AToS 24.10.2016. Za tento šprint sa nám podarilo splniť 32 SP. Najviac story pointou - 21 SP bolo za user story[ES-4] Elektrický obvod.

Pre [ES-4] Elektrický obvod bolo odhadnutých preto tak veľa SP, pretože v prípade, že by sa nepodarilo nájsť použiteľnú knižnicu na simulácie elektrického obvodu, bolo by potrebné detailne zanalyzovať ako sa elektrický obvod vnútorne správa. Zároveň by bolo potrebné vytvoriť vhodnú architektúru pre simuláciu. Tomuto a nám podarilo predísť vytvorením DLL knižnice z voľne dostupného zdroja.

1.4.2 Šprint Bentley

Šprint Bentley ktorý začal ukončením šprintu Audi 24.10.2016 bol ukončený 7.11.2016. Úlohy ktoré boli zahrnuté do toho šprintu boli odhadované na 31SP. Nakoľko sa nám nepodarilo splniť všetky úlohy bolo splnených 23SP. Nepodarilo sa nám splniť [ES-21] Grafické prvky súčiastok - napojenie na logiku.

To že sa nám nepodarilo splniť [ES-21] Grafické prvky súčiastok - napojenie na logiku bolo z dôvodu, že sme identifikovali problémy súvisiace s prácami ktoré boli vykonané v predchádzajúcom šprinte. V zásade boli dva problémy. Prvý problém bol s integráciou vytvorenej DLL knižnice do prostredia Unity. Druhý problém bol, že komponenty elektrických súčiastok v reprezentácií v Unity neboli vybavené rozlišovaním typov konektorov na jednotlivých súčiastkach.

1.4.3 Šprint Cadillac

Šprint Cadillac začal 7.11.2016 a bol úspešne ukončený 21.11.2016. V šprinte sa nám podarilo získať 39SP. Šprint považujeme za úspešný, nakoľko sa nám podarilo splniť úlohy v predpokladanej kvalite, aj keď stále evidujme námety na zlepšenia k častiam produktu, ktorý bol implementovaný v tomto

šprinte. V nasledujúcom šprinte sa pokúsime produkt rozšíriť a zapracovať námet na zlepšenie v každej module produktu.

1.4.4 Šprint Dodge

Šprint Dodge začal 21.11.2016 a bol ukončený 5.6.2016. Úlohy ktoré boli zahrnuté do toho šprintu boli odhadované na 20SP. Nakoľko sa nám nepodarilo splniť všetky úlohy bolo splnených 11SP. V šprinte sa nám nepodarilo ukoniť 2 userstory, ktoré sme pri ich analýze identifikovali ako vzáojmne prepojené a to ES-9 Create, Save a Load a ES-79 Alpha Testing I. Avšak ES-79 Alpha Testing I bol do značnej miery hotový avšak dve podúlohy zostali otvorené ktoré je potrebné vyriešiť v spolupráci s userstory ES-9 Create, Save a Load.

1.4.5 Šprint Eagle

Šprint začal 5.12.2016 a bol úspešne ukončený 12.12.2016. Tento ako jediný šprint bol naplánovaný namiesto štandardnej dĺžky šprintu ktorú máme zavedenú na 2 týždne len na jeden týždeň, Preto sme zvolili len 16SP ktoré sme sa rozhodli splniť.

1.5 Používané metodiky - referencie na dokumenty s metodikami

V projekte používame tieto metodiky:

- Práca so Source Control
- Kultúra písania kódu
- Code review
- Metodika písomnej komunikácie
- Priebeh šprintov
- Písanie dokumentácie

1.6 Stručný opis použitých metodík

1.6.1 Práca so Source Control

Základ tvorí štruktúra vetví:

1. Master vetva - je práve jedna, reprezentuje verzie produktu
2. Develop vetva - hlavný nosič vyvíjaného kódu, vetví sa z master vetvy a merguje sa do master vetvy
3. Feature vetva - reprezentuje user story, vetví sa z developu a merguje sa do developu

Nesmie sa commitovať nekompletný/neskompilovateľný kód.

Commity musia byť časté (minimálne pre každý sub-task) a dostatočne opísané.

1.6.2 Kultúra písania kódu

- kód je písaný v angličtine, odsadzovaný 4mi medzerami ("soft tabmi")
- používajú sa len explicitné dátové typy (nie var, dynamic...)

- zložené zátvorky {} sa dávajú vždy na nový riadok
- názvy tried, metód a premenných dodržiava tabuľku:

Kind	Rule
Public field	UpperCamelCase
Protected field	UpperCamelCase
Property	UpperCamelCase
Private field	_lowerCamelCase
Parameter	lowerCamelCase
Method	UpperCamelCase
Local variable	lowerCamelCase
Internal field	UpperCamelCase
Interface	IUpperCamelCase
Class	UpperCamelCase

1.6.3 Code review

Tím je rozdelený do dvoch skupín nasledovne:

- špecialisti BackEnd – Martin Cvíčela, Tomáš Liščák a Marek Matula
- špecialisti FrontEnd – Adam Blaško, Ivan Gulis a Branislav Makan

Review sa vytvára vždy pred mergovaním do develop vetvy. Revieweri sú vybraný podľa kontextu/obsahu, pod ktorý review spadá (najlepšie z priradenej skupiny).

Pre akceptovanie kódu je potrebné, aby aspoň dvaja členovia potvrdili review za splnený.

1.6.4 Metodika písomnej komunikácie

Komunikáciu rozdeľujeme podľa účastníkov komunikácie na dva typy

- interná komunikácia medzi riešiteľmi projektu
- komunikácia s verejnosťou

1.6.5 Priebeh šprintov

Backlog obsahuje všetky aktívne user story (nové User Stories sa môžu ľubovoľne pridať do Backlogu, zoradia sa na nasledovnom stretnutí).

Každá user story v backlogu musí obsahovať **názov, opis, prioritu, komponenty.**

- Do šprintu **môže** ísť iba ohodnotená User Story (**Story Points + odhad časovej náročnosti**).
- Do šprintu **nemôže** ísť User Story, ktorá nie je ohodnotená, je závislá na inú User Story zo sprintu alebo nemá splnené preddispozície v predchádzajúcich šprintoch k jej úspešnému splneniu.
- Do šprintu **musí** ísť User Story, ktorá bola priamou požiadavkou zákazníka a nič nebráni k jej úspešnému splneniu alebo je vysoko prioritná a bráni iným User Stories byť implementovné.

Každá User Story sa po naštartovaní šprintu pridelí na zodpovednosť jednému členu tímu. Aktívni členovia User Story sú povinní logovať čas (Work Log) s opisom.

Work Logy je potrebné logovať aj do User Story, nielen do Subtasku (alebo iba do User Story), kvôli Burndown Chartom Jiri.

- User Story je ukončená (môže sa dať do stavu **Done**) vtedy, ak výstup spĺňa jej funkčné požiadavky a je vhodne otestovaný s vypracovanou dokumentáciou a kompletným code review.
- Subtask je ukončený vtedy, ak výstup spĺňa jej funkčné požiadavky, je vhodne otestovaný a je k nemu vypracovaná dokumentácia.
- Šprint sa chápe za ukončený, ak vypršalo 14 dní a vykonala sa retrospektíva šprintu.
- Šprint sa chápe za úspešne ukončený, ak všetky User Story v ňom sú uzatvorené, majú k vytvorenú dokumentáciu a výstupy sú zmergované.

1.6.6 Písanie dokumentácie

Dokumentácia má stanovenú štruktúru a členenie kapitol.

Na inžinierskom diele pracuje každý na svojej časti osobitne.

Pre publikáciu výsledku má každý k dispozícii jednu page v Confluence.

Výsledok sa spojí v editore Word.

Zápisky z cvičení vždy zaznamenáva jedna osoba, ktorá následne pošle tieto zapísané informácie manažérovi dokumentácie(Tomáš Liščák), ktorý ich spracuje podľa výstupnej šablóny a pošle ich manažérovi webu (Branislav Makan) na vyvesenie na stránku.

O robenie retrospektívy šprintov sa stará náš scrum master (Adam Blaško).

1.7 Globálna retrospektíva ZS/LS

1.7.1 Zimný semester

Počas zimného semestra sa šprint od špritu razantne menila podoba práce ľudí v tíme. Objektívne sa však dá povedať, že k lepšiemu. Počas prvej polovice prvého šprintu sme ako tím ešte nemali dostupné všetky nástroje na podporu vývoja - manažovací nástroj, source control, nástroj na code review ani tímovú wiki.

1.7.2 Čo sme robili dobre

- **Riadenie sa SCRUM.** Už od začiatku projektu sme sa čo najlepšie všetci snažili dodržiavať základy SCRUM metodiky - planning poker, hodnotenie story pointami, rozdelenie projektu na user sotires, vytvorenie produktového backlogu.
- **Zdielanie vedomostí.** Prirodzene každý člen tímu má iné vedomosti a skúsenosti. Veľmi pozitívne však hodnotíme to, že o vedomosti sa členovia hneď podelili - napr. išlo o skúsenosti s prácou s Gitom, skúsenosti s použitými technológiami a podobné

1.7.3 Čo sme zlepšili

- **Code review.** Prehliadky kódu sme začali robiť až v druhom šprinte, pričom sa robili na báze toho, že prehliadky mohol robiť každý každému. To malo za následok stav, keď nebolo jasne definované, že sa prehliadka skončila, pretože sme nemali definovaný stav ukončenia prehliadky. V treťom šprinte sme mali definovanú metodika robenia code review, ktorá zahŕňa všetky podmienky vytvárania prehliadok, samotného prehliadania a aj ukončenia prehliadok. Tým sme dĺžku trvania jednej prehliadky za ideálnych podmienok okresali na približne 30 minút. Počas druhého šprintu prehliadky trvali aj 8 hodín.
- **Písanie zdrojových kódov.** Vytvorili sme metodiku, ktorá rozširuje základné pravidlá písania zdrojového kódu v jazyku C#. Dodržiavanie tejto metodiky sa kontroluje pri prehliadkach zdrojového kódu, čo výrazne zlepšilo kvalitu kódu v celom projekte.
- **Definition of done.** Vytvorili sme globálnu definition of done pre všetky user stories. Táto DoD zahŕňa kroky, ktoré je potrebné vykonať k úspešnému uzatvoreniu úlohy, čo výrazne pomohlo k zníženiu doby uzatvárania úlohy, pretože každý člen tímu presne vie povedať, či je jeho úloha splnená alebo nie.

1.8 Motivačný dokument

1.8.1 Predstavenie Tímu

Všetci sme čerstvými absolventami bakalárskeho štúdia na FIIT STU. Študovali sme v odbore informatika s trvaním 3 roky. So štúdiom sme nemali veľké problémy, nakoľko ani jeden z nás nemusel opakovať žiaden predmet. Nemáme preto probém s procedurálnymi ani objektovo orientovanými jazykmi, a nie je nám cudzí ani web. Sme ochotní sa učiť nové veci a rozvíjať naše znalosti.

1.8.2 Skúsenosti získané v škole

Znalosti získané počas doterajšieho štúdia sú hlavne definované výberom témy bakalárskej práce. Z tohoto pohľadu máme ako tím skúsenosti s analýzou problémov a dát, strojovým učením, Big Data, umelou inteligenciou, dátovými štruktúrami, paralelizáciou, optimalizáciou algoritmov a kódu, bezpečnosťou, bezdrôtovou komunikáciou a mobilnými aplikáciami.

1.8.3 Oblasti záujmu

Na inžinierskom štúdiu máme člena v softvérovom inžinierstve, kým ostatní študujeme informačné systémy. Výber nepovinných predmetov odzrkadľuje naše záujmy. Máme ľudí pohybujúcich sa v počítačovej grafike, dátovej analýze, návrhu a vývoji softvérových systémov a počítačovej bezpečnosti, čo potvrdzuje našu všestrannosť. Mnohí z nás máme záujem aj o vývoj video hier a je to potenciálna oblasť, ktorou by sme sa v budúcnosti zaoberali.

1.8.4 Mimoškolské skúsenosti

Na základe väčšinou pracovných mimoškolských aktivít máme ako tím skúsenosti s vývojom mobilných aplikácií aj so serverovým back-endom, vývojom podnikového softvéru, hardvérom, databázami, automatizáciou a verziovacím systémom - napr. git. Väčšina členov tímu aj aktuálne pracuje v oblasti informatiky.

1.8.5 Motivácia k téme [EduSim]

Klasické prezentácie často nedokážu pripútať a udržať pozornosť študenta. Preto dnes existujú aj rôzne iné prístupy ovplyvnené prvkami hier. Vhodná ukážka úspešnosti hier v edukatívnej sfére je aj v nasledovnom článku: https://goo.gl/ucPL26. Skrátka, hráči dokázali nájsť doposiaľ neznámy proteín, ktorý by vedel pomôcť pri liečbe Alzeihmerovej choroby. Určite vynaložíme dodatočné úsilie aj na optimalizáciu webovej aplikácie aj na mobilných zariadeniach s dotykovou obrazovkou, ktoré sú v dnešnej dobe veľmi populárne. Na tento účel existuje veľké množstvo frameworkov, ktoré by sme použili a tak si uľahčili prácu na front-ende (napr. v súčasnosti populárne Twitter Bootstrap alebo Google Material Design sú optimalizované aj na dotykové obrazovky). Do aplikácii by sme teda zakomponovali rôzne herné prvky, ktorými by sme používateľov aj pobavili - edukácia môže byť aj zábavná. Týmto myslíme hlavne na rôzne hádanky, puzzle pre mladších, bodové hodnotenie a prípadné testy a súťaže. Ľudia veľmi radi súťažia a veríme, že takéto herné prvky aplikácii určite vo veľkej miere pomohnú. Z druhej strany, radi by sme aj učiteľom umožnili tvoriť pomôcky na prednášky v tvare grafických obrázkov, krátkych animácií alebo úloh, ktoré by žiaci, či študenti riešili. V našom tíme sa už každý stretol s webovými technológiami v menšom alebo väčšom množstve. Niektorí preferujú programovanie back-endu, kým iní zase front- end čo nám umožní ľahké rozdelenie úloh. V tíme máme aj ľudí, ktorý už majú za sebou nejakú tú webovú aplikáciu v praxi, teda mimo školy. Týmto projektom by sme chceli priblížiť už veľmi úspešný a rozsiahly herný priemysel k edukačnej sfére.

1.8.6 Motivácia k téme [eMotion]

Produkt prieniku viacerých disciplín je väčšinou zaujímavá a užitočná záležitosť, ktorá aj v tomto prípade pomôže automatizovať a doplniť prácu pracovníkom v medicíne. Automatizované merania a ich analýza pomáhajú korektnosti, rýchlosti a tým pádom znižovaním záťaže a skvalitňovaním času týmto pracovníkom. Samotná registrácia emócii je téma s potenciálom, pričom sa dá experimentovať s rôznymi metódami analýzy meraných dát pomocou strojového učenia, ktoré sa práve pre big data z meračov stávajú čoraz účinnejšie. Jedná sa aj o samotnú pomoc ľuďom, čo je samo o sebe motivujúce a inšpirujúce. Toto všetko sú dôvody nášho tímu pre záujem o túto tému. Vieme využiť naše skúsenosti so strojovým učením, ovládame objektovo- orientované jazyky, ktoré téma vyžaduje. Taktiež máme skúsenosti s vývojom mobilných aplikácií a analýzou dát.

1.9 Metodiky

1.9.1 Práca so Source Control

Kód commitovaný do repozitára

V repozitáry sa nachádza iba funkčný kód, ktorý podlieha metodike kultúry písania kódu. Nesmie sa commitovať kód, ktorý je neúplný, nekompletný a hlavne

sa **nesmie** commitovať **neskopilovateľný** kód. Zdrojové súbory nesmú obsahovať zakomentovaný kód.

Commity

Každý commit je dostatočne dobre opísaný, ideálne v anlgickom jazyku. Commit by sa mal robiť po každom pridaní kompletnej a ucelenej logickej časti user story, minimálne pre každý sub-task danej story. Ak je to možné tak popis commitu by mal obsahovať označenie subtasku z Jiry.

Vetvenie kódu

Repozitár obsahuje jednu vetvu *master*, jednu vetvu *develop* a niekoľko *feature* vetiev.

Master vetva

Repozitár obsahuje práve jednu hlavnú vetvu - *master*, ktorá je odrazom najnovšej verzie produktu, ktorá je momentálne v produkcií. To znamená, že vo vetve *master* sa okrem zlučovaní (merge) nenachádzajú priamo žiadne commity. Z vetvy *master* sa na začiatku vývoja vetví vetva *develop*, a iba tá sa následne pripája naspať do *master* vetvy. *Develop* by sa mal do *master* pripájať vždy pri vydávaní novej verzie produktu - na konci každého jedného šprintu.

Develop vetva

Vetva *develop* predstavuje hlavného nosiča vyvíjaného kódu. Vetvia sa z nej *feature* vetvy, a commity v samotnej *develop* vetve predstavujú zmeny súvisiace s integráciou jednotlivých *feature* vetiev.

Feature vetvy

Každá *feature* vetva predstavuje jednu user story, ktorá je vyvíjaná. Ich názov zodpovedá schéme **feature/<názov user story z JIRA>**, napr. **feature/ES-1-gui-graficka-cast**. Pre vytváranie týchto vetiev sa odporúča použiť Jiru. *Feature* vetvy sa vetvia z vetvy *develop* a do tejto vetvy sa po ukončení vývoja aj zlúčia (merge).

Zlučovanie vetví

Pre zlučovanie vetiev je nutné splniť podmienky nižšie definované podmienky.

Feature → develop

User story, pre ktorú bola vetva vytvorená musí byť splnená podľa definition of done danej story. Kód musí byť otestovaný a zrevidovaný.

Develop → master

Všetky user story z daného šprintu sú hotové a je potrebné vytvoriť release verziu produktu.

1.10 Kultúra písania kódu

V rámi projektu pracujeme s grafickým enginom Unity. V rámci tohto grafického enginu sa používajú skripty buď v jazyku JavaScript alebo C#. Ak je to možné, tak je potrebné sa vyhýbať JavaScript-u a všetky skripty písať v C#. Všetky nasledujúce pravidlá je dôležité dodržiavať pri písaní kódu v jazyku C#, ale ak je to možné, tak aj na prvy vlastné grafickému enginu Unity - napríklad názvy súborov (assetov, scén, atď...). Všetky použité konvencie plynú zo zaužívaných konvencií písania C# kódu. Ak metodika nepokrýva nejakú sféru písania kódu, je potrebné konzultovať zdroje dostupné na webe:

- http://www.c-sharpcorner.com/UploadFile/8a67c0/C-Sharp-coding-standards-and-naming-conventions/
- https://msdn.microsoft.com/en-us/library/ff926074.aspx

Pre prácu s Visual Studiom je vhodné naištalovať plugin ReSharper, ktorého licencia je súčasťou študenstkého balíka spolu s ostatnými IDE - je potrebné iba zadať číslo ISIC-u. Tento plugin kontroluje kód a pomáha dodržiavať všetky konvencie písania C# kódu. Jediná zmena oproti našim štandardom je tá, že sa prioritne znaží používať implicitné dátové typy (var) - to je potrebné v nastaveniach pluginu vypnúť. Inak vás bude plugin upozorňovať na zlé názvy metód, tried, premenných a bude poskytovať inteligentnú automatickú opravu. Toto nastavenie najdete cez ReSharper→ Options→ Code Editing→ C#→ Code Style kde pre prve 3 nastavenia ('var' usage in declaration) nastavte moznost Use explicit types.

1.10.1 Jazyk

Jazyk používaný pri písaní kódu je angličtina. To platí aj pre názvy projektov, súborov a zložiek. Slovenčina sa môže objaviť jedine pri písaní komentárov a v súboroch používaných pre lokalizáciu (resource files).

1.10.2 Odsadzovanie textu

Kód je odsadzovaný 4 medzerami. Toto je default nastavenie IDE Visual studio. Nastavenie sa dá zmeniť v Tools→ Options→ Text Editor→ C#→ Tabs. Je potrebné mať nastavené Tab size = 4, Indent size = 4 a zvolenú možnosť Insert spaces.

1.10.3 Dátové typy

Je zakázané používať implicitné dátové typy (var, dynamic). Používajú sa iba explicitné dátové typy.

1.10.4 Názvy

Názvy tried, metód a premenných sú opisné. Výnimkou sú parametre pre krátke lambda výrazy - krátke znamená najviac 5 riadkov kódu. Je zakázané používať skratky. Výnimkou sú dobre známe skratky, ako napríklad HTML, FTP a podobné.

Použitá veľkosť písma (case) je definovaná nasledujúcou tabuľkou:

Kind	Rule
Private field	_lowerCamelCase
Public field	UpperCamelCase
Protected field	UpperCamelCase
Internal field	UpperCamelCase
Property	UpperCamelCase

Method	UpperCamelCase
Class	UpperCamelCase
Interface	IUpperCamelCase
Local variable	lowerCamelCase
Parameter	lowerCamelCase

1.10.5 Zátvorky

Zložené zátvorky {} sa dávajú vždy na nový riadok. Používajú sa pre každý blok textu, takže aj pre jeden riadok kódu.

Podmienky

```
if (true)
{
        OneLineIf();
}

if ()
{
        FirstLine();
        SecondLine();
}
```

1.10.6 Návraty metód

Ak je to možné, tak každá metóda má maximálne jeden návratový bod (return;).

Návrat metód

```
    private bool ThisIsIncorrect()
    {
        if (true)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    private bool ThisIsCorrect()
    {
        bool result;
        if (true)
        {
            result = true;
        }
        else
```

```
        {
            result = false;
        }
        return result;
    }
```

Na manažment zdrojových kódov sa používa Git, konkrétne Bitbucket. Ako branching model sa používa Git Flow. Na prácu s git repozitárom sa odporúča používať Sourcetree

1.11 *Metodika písomnej komunikácie*

Komunikácia je rozdelená do dvoch oblastí na základe členov komunikácie.

- interná komunikácia medzi riešiteľmi projektu
- komunikácia s verejnosťou

1.11.1 Komunikácia medzi riešiteľmi projektu

Na písomnú komunikáciu medzi riešiteľmi tímu je uprednostňovaný nástroj HipChat od spoločnosti Attlassian.

Komunikácia je organizovaná do takzvaných miestností (angl. room).

Ak v komunikácií je potrebné vyzvať osobu na zapojenie sa do diskusie, je potrebné použiť prepojenie. Prepojenie sa vytvára ak používateľské meno je napísané so znakom @ [zavináč]. Pokiaľ komunikujúci potrebuje vyzvať všetkých členov danej komunikačnej miestností, je potrebné napísať @all.

1.11.2 Komunikácia s verejnosťou

Pri komunikácií s verejnosťou je potrebné dbať na slušnosť, vecnosť, korektnosť a pozitívnu reprezentáciu tímu.

Pred komunikáciu je potrebné si potvrdiť správnosť svojich tvrdení u ostatných členov tímu, buď v prítomnosti všetkých členov tímu, alebo písomne v nástroji HipChat.

1.12 *Priebeh šprintov*

1.12.1 Backlog

Backlog obsahuje všetky aktívne user story.

1.12.2 Tvorba nových User Stories

Nové User Stories sa môžu ľubovoľne pridať do Backlogu. Povinné polia musia byť vyplnené pri ich vnášaní. Tieto sú uvedené nižšie.

1.12.3 Obsah User Stories

Každá user story v backlogu musí obsahovať:

- **názov** - vyznačuje o čo v danej User Story ide;
- **opis** - pre user story sa pridá hneď pri jej vytvorení. Tento popis môže byť modifikovaný v budúcnosti;

- **prioritu** - priorita user story sa rozhodne na najbližšom stretnutí;
- **komponenty** - vyznačujú akých oblastí sa User Story týka. Komponenty sú preddefinované.

1.12.4 Zoradenie User Stories v Backlogu

User Stories budú vždy zoradené podľa ich priority. V prípade, že User Story bola pridaná mimo stretnutia, táto User Story bude zaradená podľa priority na nasledovnom stretnutí.

1.12.5 Voľba User Stories pre šprint

Do šprintu sa koná výber z aktívnych User Stories v Backlogu. Výber pozostáva z viacerých fáz.

1.12.6 Ohodnotenie User Stories

Do šprintu **môže** ísť iba ohodnotená User Story. Hodnotia sa dve veci:

- **Story Points** - pomocou scrum pokeru;
- **odhad časovej náročnosti** User Story.

1.12.7 Scrum poker

Scrum poker prebieha nasledovne:

1. Každý člen vyjadrí svoju predstavu náročnosti pomocou kartičky. Kartičku si pripraví vopred tak, aby ju žiaden iný člen tímu nevidel predtým než ju odhalí.
2. Ak sú hodnoty všetkých kartičiek rovnaké, hodnotenie končí a User Story nadobúda Story Points rovné číslu na kartičkách.
3. Členovia s najmenším a najväčším odhadom náročnosti oddôvodnia svoju voľbu.
4. K diskúsii sa pripoja aj iný členovia a oddôvodnia prečo je ich odhad iný.
5. Návrat na prvý krok.

1.12.8 Odhad časovej náročnosti

Odhad časovej náročnosti sa koná voľnejšie. Členovia, ktorí majú v danej oblasti skúsenosť odhadnú koľko človekohodín daná User Story asi zaberie a priemerná hodnota týchto odhadov sa berie ako časové ohodnotenie User Story.

1.12.9 Výber do šprintu

Do šprintu **nemôže** ísť User Story, ktorá:

- nie je ohodnotená;
- je závislá na inú User Story, ktorá sa berie do šprintu*;
- nemá splnené preddispozície v predchádzajúcich šprintoch k jej úspešnému splneniu*.

*vínimkou je, keď sa nedokončená User Story, na ktorej je iná User Story závislá, prenáša do ďalšieho šprintu.

Do šprintu **musí** ísť User Story, ktorá:

- bola priamou požiadavkou zákazníka a nič nebráni k jej úspešnému splneniu;
- je vysoko prioritná a bráni iným User Stories byť implementovné.

1.12.10 Priebeh šprintu

Každá User Story sa po naštartovaní šprintu pridelí na zodpovednosť jednému členu tímu. Táto osoba a iné osoby, ktoré na danej User Story budú pracovať, sú povinní prideliť si a vyhodotiť časové odhady Subtaskov, ktoré sú už zadefinované. Títo aktívni členovia pre danú User Story majú právo vytvárať nové Subtasky s povinnosťou časovo ich vyhodnotiť, pridať k ním opis a prideliť si ich. Aktívni členovia User Story sú povinní logovať čas (Work Log), ktorí strávili prácou nad User Story. Tieto logy budú vytvárať aj s vhodným popisom.

1.12.11 Ukončenie User Story

Ukončenie User Story musí v prvom rade spĺňať **Definition of Done**. Po jeho splnení môže byť User Story uzatvorená (stav Done).

Subtasky musia spĺňať **Definition of Done, časť pre Subtasky**. Tieto kritériá sú voľnejšie.

1.12.12 Definition of Done

User Story je ukončená (môže sa dať do stavu **Done**) vtedy, keď spĺňa nasledovné podmienky:

- výstup spĺňa jej funkčné požiadavky;
- výstup je vhodne otestovaný;
- k nej je vypracovaná dokumentácia;
- nad jej výstupom je vykonaný code review.

Subtask je ukončený vtedy, keď spĺňa nasledovné podmienky:

- výstup spĺňa jej funkčné požiadavky;
- výstup je vhodne otestovaný;
- k nemu je vypracovaná dokumentácia.

1.12.13 Ukončenie šprintu

Šprint sa chápe za ukončený, keď spĺňa nasledovné podmienky:

- vypršalo 14 dní;
- vykonala sa retrospektíva šprintu.

Šprint sa chápe za úspešne ukončený, keď spĺňa nasledovné podmienky:

- všetky User Story v ňom sú uzatvorené;
- všetky User Story majú k ním vytvorenú dokumentáciu;
- výstupy všetkých User Stories sú zmergované.

1.13 *Písanie dokumentácie*

1.13.1 Členenie

Dokumentácia má pozostávať z dvoch hlavných dokumentov:

- i**nžinierske dielo**

- **dokumentácia k riadeniu**.

1.13.2 Inžinierske dielo

Má pozostávať z kapitol:

- úvod
- globálne ciele pre jednotlivé semestre
- celkový pohľad na systém
- moduly systému
- inštalačná príručka
- technická dokumentácia

Celkový pohľad na systém má mať v sebe:

- model architektúry
- dátový model
- diagram modulov
- diagram tried
- zoznam priložených elektronických dokumentov

Moduly systému majú byť v dokumentácii členené za sebou, pričom každý z modulov bude pozostávať z jednotlivých príbehov šprintov, v ktorých sa na ňom robilo. Každý príbeh šprintu predstavuje istý ucelený prírastok, ktorý má byť vždy členený do jednotlivých fáz vývoja, ako sú: analýza, návrh, implementácia a testovanie. Nie každý príbeh šprintu musí obsahovať všetky fázy vývoja, avšak tento jav je žiadaný, nakoľko má každý príbeh reprezentovať jeden menší projekt sám o sebe.

1.13.3 Dokumentácia k riadeniu

- úvod
- role členov tímu a podiel práce
- aplikácie manažmentov
- sumarizácie šprintov
- používané metodiky
- globálna retrospektíva za semester
- motivačný dokument
- export evidencie úloh z jiry

Role členov tímu a podiel práce

Opísanie rôl, zodpovedností a z toho vyplývajúcich manažérskych činností každého člena tímu, ako aj napr. podiel práce na dokumentácii.

Aplikácie manažmentov

Opis všeobecných tímových činností riadenia projektu.

Sumarizácie šprintov

Retrospektíva šprintov

Používané metodiky

Metodiky v confluence, ktorými sa projekt riadil. Takisto je treba pridať metodiky z MIS vychádzajúce z role každého člena tímu, ktoré sú aplikovateľné na projekt.

1.13.4 Workflow

Na inžinierskom diele pracuje každý na svojej časti osobitne, s tým, že je jeho časť vždy tie časti systému, na ktorých pracoval. Pre zhotovovanie svojej časti má každý k dispozícii jednu page v confluence, pričom je takto zabezpečená simultálna práca bez prepisovania si textov. Po každom šprinte je si každý člen tímu povinný takto zdokumentovať svoj príspevok do projektu.

Niektoré ďalšie časti dokumentácie, najmä časti týkajúce sa dokumentácii k riadeniu, sa dopíšu v závislosti na tom, kto si vezme ktoré časti na starosti. Tieto časti sa však dopisujú vždy v prípade ich potreby, kedy vznikne požiadavka od niektorého člena tímu na dopísanie týchto častí a následne sa vyberie člen tímu, alebo prihlási dobrovoľník z tímu, ktorý sa zaviaže dopracovať tieto časti do určitého termínu. Pokiaľ však nevznikne potreba týchto častí od nikoho z tímu, tak táto potreba vznikne automaticky u všetkých nespravených častí tejto dokumentácie týždeň pred odovzdaním dokumentácie. Táto potreba sa vyrieši úplne rovnako, ako potreba od ktoréhokoľvek člena tímu.

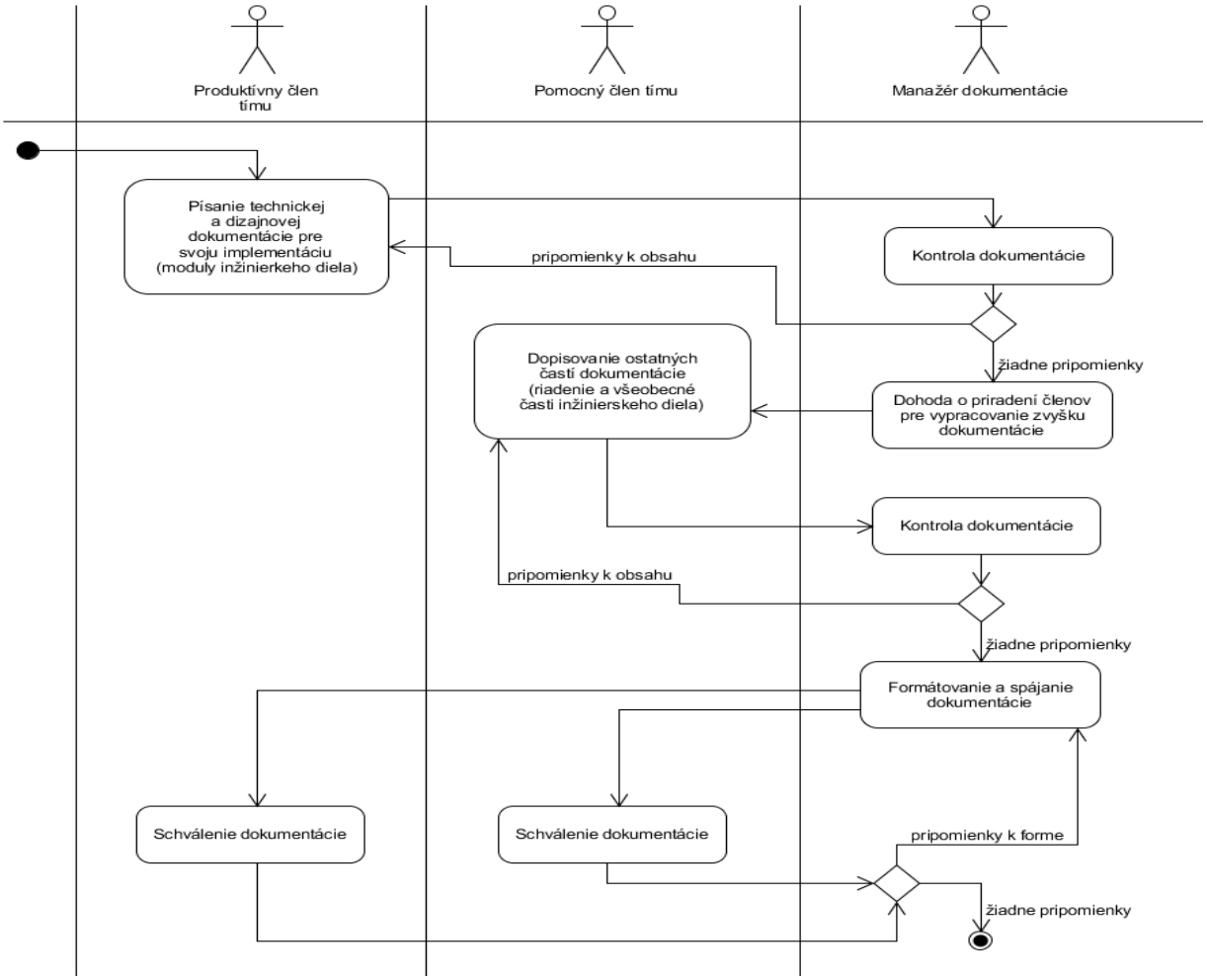
Po vyhotovení týchto častí, pred bodom odovzdania dokumentácie(Tomáš Liščák), výsledok manažér dokumentácie spojí a skontroluje v editore Word. Následne dokumentáciu odovzdá do systému AIS.

Zápisky z cvičení vždy zaznamenáva jedna osoba, ktorá následne pošle tieto zapísané informácie manažérovi dokumentácie(Tomáš Liščák), ktorý ich spracuje podľa výstupnej šablóny a pošle ich manažérovi webu (Branislav Makan) na vyvesenie na stránku.

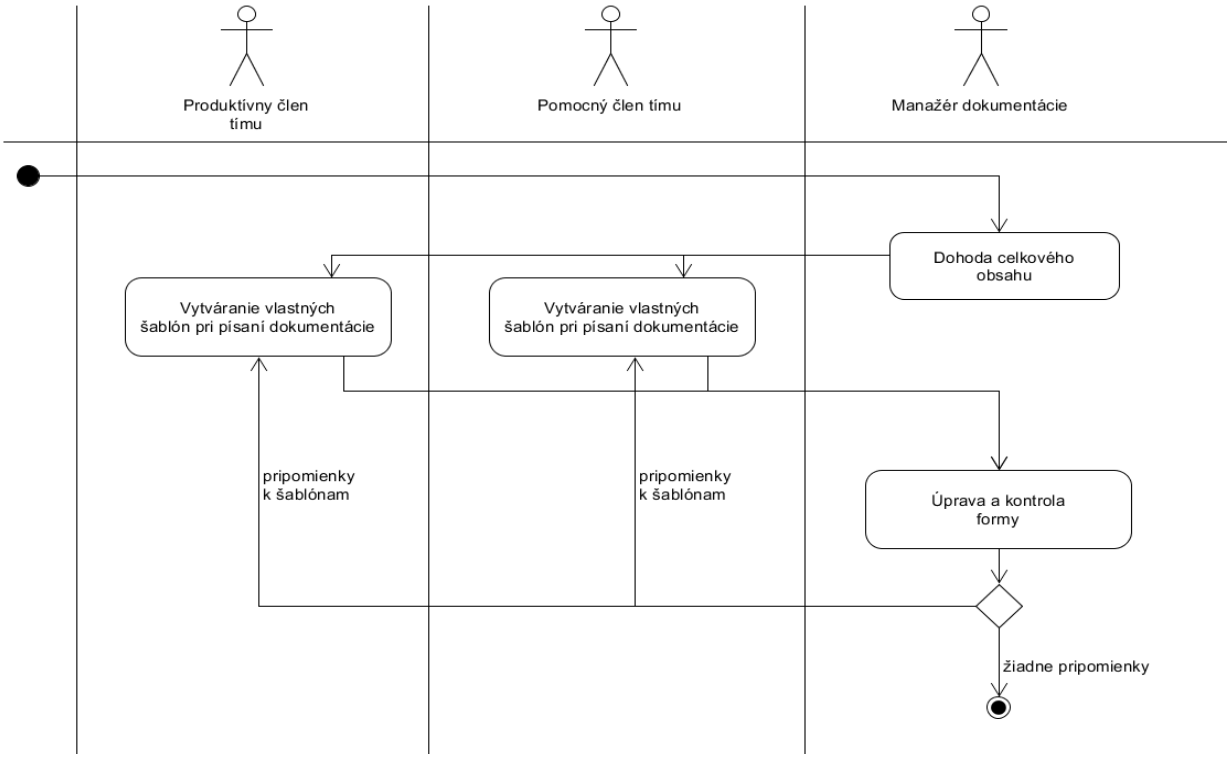
Retrospektíva zo šprintov sa vykonáva na stretnutiach, kedy sa konzultuje posledný šprint. Robí sa ako súčasť funkcionality JIRA a preto aj zostáva v nej uložená.

Workflow tým pádom obsahuje 3 procesy manažmentu dokumentácie:

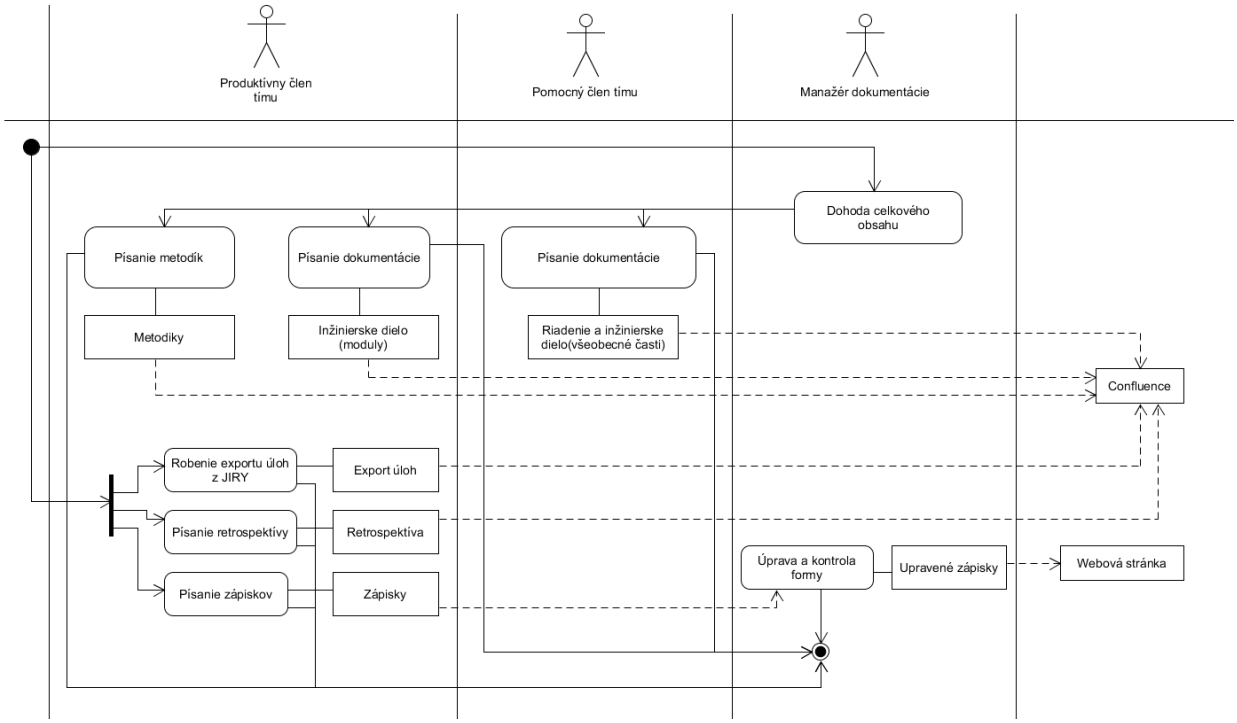
Proces tvorenia dokumentácie



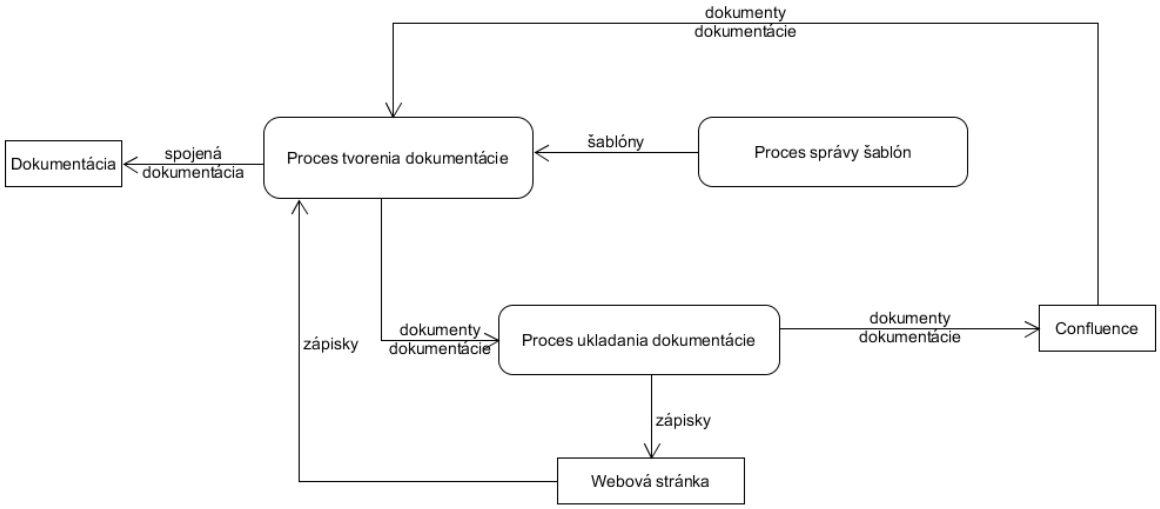
Proces tvorenia šablón



Proces správy dokumentácie



Spoločný dátový diagram procesov



1.14 *Code review*

1.14.1 Rozdelenie kompetencií k reviewovaniu

Členovia projektu sa z hľadiska oblasti do ktorej prevažne prispievajú pri vývoji aplikácií rozdeľujú do dvoch skupín

- špecialisti BackEnd
- špecialisti na FrontEnd

Rozdelenie jednotlivých členov tímu sme si zvolili nasledovne:

špecialisti BackEnd – Martin Cvíčela, Tomáš Liščák a Marek Matula

špecialisti FrontEnd – Adam Blaško, Ivan Gulis a Branislav Makan

1.14.2 Výber členov na review

Výber sa deje podľa možností v takom poradí ako je uvedené v tomto texte.

Podľa toho do ktorej oblasti vývoj v danej vetve spadá, tak z danej oblasti sú vybratí členovia na review. Ak nie je možné vybrať dostatočný počet členov z oblasti do ktorej review primárne spadá, vyberieme členov z inej oblasti tak aby mali čo najbližšie podľa možností ku oblasti ktorú budú reviewovat.

1.14.3 Vytváranie codereview v nástroji Crucible

Codereview je potrebné vytvoriť vždy ako sa ktokoľvek rozhodne pre merge do develope vetvy. Codereview sa považuje sa splnené ak je aspoň od dvoch reviewerov zdrojový kód skontrolovaný a zároveň potvrdený, že zmeny ktore požadoval boli zapracované alebo vydiskutované a tým vyriešené.

Každý splnený codereview má nula nevyriešených požiadaviek.

1.15 Export úloh šprintov

1.15.1 Šprint Audi

[ES-1] GUI - grafická časť Created: 12/Oct/16 Updated: 21/Oct/16 Resolved: 21/Oct/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Highest
Reporter:	Branislav Makan	**Assignee:**	Ivan Gulis
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	0 minutes	**Remaining Estimate:**	0 minutes
Σ Time Spent:	1 day, 1 hour	**Time Spent:**	0 minutes
Σ Original Estimate:	7 hours	**Original Estimate:**	0 minutes

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-22	Hlavné menu	Sub-task	Done	Ivan Gulis
	ES-23	Toolbar	Sub-task	Done	Ivan Gulis

	ES-24	Toolbox	Sub-task	Done	Ivan Gulis
	ES-25	Properties	Sub-task	Done	Ivan Gulis
	ES-26	Explorer	Sub-task	Done	Ivan Gulis
	ES-27	Error log	Sub-task	Done	Ivan Gulis
Sprint:	Sprint I - Audi				
Story Points:	3				

Návrh a implementácia grafickej časti používateľského rozhrania - okná, menu a podobne.

[ES-2] Pracovná plocha Created: 12/Oct/16 Updated: 24/Oct/16 Resolved: 24/Oct/16	
Status:	Done
Project:	EduSim
Component/s:	Core, GUI
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	High
Reporter:	Branislav Makan	**Assignee:**	Branislav Makan
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	30 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	1 day, 6 hours	**Time Spent:**	Not Specified
Σ Original Estimate:	1 day, 2 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-28	Grid	Sub-task	Done	Branislav Makan
	ES-29	Zoom	Sub-task	Done	Branislav Makan
	ES-30	Create	Sub-task	Done	Branislav Makan
	ES-31	Move	Sub-task	Done	Branislav Makan
	ES-32	Rotate	Sub-task	Done	Branislav Makan
Sprint:	Sprint I - Audi				
Story Points:	5				

Návrh a implementácia pracovnej plochy - použitie mriežky na úzadí.

[ES-3] Súčiastky Created: 12/Oct/16 Updated: 24/Oct/16 Resolved: 24/Oct/16	
Status:	Done
Project:	EduSim
Component/s:	Core
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	High
Reporter:	Branislav Makan	**Assignee:**	Tomas Liscak
Resolution:	Done	**Votes:**	0
Labels:	Core		
Σ Remaining Estimate:	1 hour, 15 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	5 hours, 45 minutes	**Time Spent:**	Not Specified

Σ Original Estimate:	7 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-34	Zoznam súčiastok elektrického obvodu	Sub-task	Done	Tomas Liscak
	ES-33	Generické	Sub-task	Done	Tomas Liscak
	ES-35	Vlastnosti súčiastok	Sub-task	Done	Tomas Liscak
Sprint:	Sprint I - Audi				
Story Points:	3				

Návrh a implementácia generických a súčiastok elektrického obvodu.

[ES-4] Elektrický obvod Created: 12/Oct/16 Updated: 24/Oct/16 Resolved: 24/Oct/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Adam Blaško
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining	2 hours, 30 minutes	**Remaining**	Not Specified

Estimate:		**Estimate:**	
Σ Time Spent:	1 day, 2 hours, 30 minutes	**Time Spent:**	Not Specified
Σ Original Estimate:	1 day, 2 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-36	Vlastnosti elektrického obvodu	Sub-task	Done	Martin Cvicela
	ES-37	Dátové štruktúry	Sub-task	Done	Adam Blaško
	ES-38	Implemenácia logiky	Sub-task	Done	Marek Matula
Sprint:	Sprint I - Audi				
Story Points:	21				

Návrh implementácie a dátových štruktúr k elektrickému obvodu.

1.15.2 Šprint Bentley

[ES-6] GUI - logická časť I Created: 12/Oct/16 Updated: 06/Nov/16 Resolved: 06/Nov/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Ivan Gulis

Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	0 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	3 days, 1 hour	**Time Spent:**	Not Specified
Σ Original Estimate:	1 day, 3 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-48	Toolbox	Sub-task	Done	Ivan Gulis
	ES-49	Merge výsledkov z Audiho	Sub-task	Done	Ivan Gulis
	ES-51	Debug okno	Sub-task	Done	Ivan Gulis
	ES-54	Scrollovanie	Sub-task	Done	Ivan Gulis
Sprint:	Sprint II - Bentley				
Story Points:	8				

Implementácia funkcionality GUI - toolbox a debug log. Mergnutie výsledkov z predchádzajúceho sprintu.

[ES-10] Lokalizácia - resource files Created: 12/Oct/16 Updated: 07/Nov/16 Resolved: 07/Nov/16

Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Adam Blaško
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	1 hour	**Remaining Estimate:**	Not Specified
Σ Time Spent:	2 hours	**Time Spent:**	Not Specified
Σ Original Estimate:	3 hours	**Original Estimate:**	Not Specified

Sub-Tasks:

Key	Summary	Type	Status	Assignee
ES-60	Studium pouzitia resource suborov v u...	Sub-task	Done	Adam Blaško
ES-61	Vytvorenie resource suborov	Sub-task	Done	Adam Blaško
ES-62	Pouzivanie resource suborov v existuj...	Sub-task	Done	Adam Blaško
ES-63	Prepinanie lokalizacii	Sub-task	Done	Adam Blaško

Sprint:	Sprint II - Bentley
Story Points:	2

Vytvorenie resource filov a ich implementácia do kodu.

[ES-40] Vykreslovanie čiar Created: 24/Oct/16 Updated: 07/Nov/16 Resolved: 07/Nov/16

Status:	Done
Project:	EduSim

Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Martin Cvicela
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	1 hour	**Remaining Estimate:**	Not Specified
Σ Time Spent:	1 day, 6 hours	**Time Spent:**	Not Specified
Σ Original Estimate:	1 day, 3 hours, 30 minutes	**Original Estimate:**	Not Specified

Sub-Tasks:

Key	Summary	Type	Status	Assignee
ES-41	Grafické vykreslenie čiar medzi dvomi...	Sub-task	Done	Martin Cvicela
ES-42	Pripájanie k súčiastkam	Sub-task	Done	Martin Cvicela
ES-43	Pridanie predchodcu a nasledovníka k ...	Sub-task	Done	Martin Cvicela

Sprint:	Sprint II - Bentley
Story Points:	5

[ES-52] Vylepšenie pracovnej plochy Created: 24/Oct/16 Updated: 12/Nov/16 Resolved: 06/Nov/16

Status:	Done
Project:	EduSim
Component/s:	Core, GUI
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Branislav Makan
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	5 hours, 10 minutes	**Remaining Estimate:**	0 minutes
Σ Time Spent:	5 hours, 10 minutes	**Time Spent:**	50 minutes
Σ Original Estimate:	1 day, 2 hours, 30 minutes	**Original Estimate:**	Not Specified

Sub-Tasks:

Key	Summary	Type	Status	Assignee
ES-55	Grid upgrade	Sub-task	Done	Branislav Makan
ES-56	Posun kamery	Sub-task	Done	Branislav Makan
ES-57	Delete	Sub-task	Done	Branislav Makan
ES-58	Prekrývanie - zakázať	Sub-task	Done	Branislav Makan
ES-59	Deselect	Sub-task	Done	Branislav Makan
ES-64	Background	Sub-task	Done	Branislav Makan

Sprint:	Sprint II - Bentley

Story Points:	8

1.15.3 Šprint Cadillac

[ES-14] Simulácia el. obvodu v grafickom prostredí Created: 12/Oct/16 Updated: 21/Nov/16 Resolved: 21/Nov/16

Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Tomas Liscak
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	1 day, 6 hours, 30 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	1 day, 1 hour, 30 minutes	**Time Spent:**	Not Specified
Σ Original Estimate:	2 days, 5 hours	**Original Estimate:**	Not Specified

Sub-Tasks:

Key	Summary	Type	Status	Assignee
ES-65	Vypis zakladnych vlastnosti obvodu	Sub-task	Done	Marek Matula
ES-72	Otestovať algoritmus	Sub-task	Done	Tomas Liscak

	ES-75	Spustenie simulácie	Sub-task	Done	Tomas Liscak
Sprint:	Sprint III - Cadillac				
Story Points:	8				

[ES-21] Grafické prvky súčiastok - napojenie na logiku Created: 12/Oct/16 Updated: 19/Nov/16 Resolved: 19/Nov/16

Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Marek Matula
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	0 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	2 days, 3 hours, 10 minutes	**Time Spent:**	Not Specified
Σ Original Estimate:	2 days, 3 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-44	Mapovanie DLL objektov na C#	Sub-task	Done	Marek Matula

		objekty			
	ES-45	Riešenie logických prepojení	Sub-task	Done	Tomas Liscak
	ES-46	Funkčná simulácia	Sub-task	Done	Marek Matula
Sprint:	Sprint II - Bentley, Sprint III - Cadillac				
Story Points:	8				

[ES-53] Properties - Get/Set atribútov súčiastok Created: 24/Oct/16 Updated: 18/Nov/16 Resolved: 18/Nov/16

Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	High
Reporter:	Branislav Makan	**Assignee:**	Branislav Makan
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	1 hour	**Remaining Estimate:**	Not Specified
Σ Time Spent:	7 hours, 30 minutes	**Time Spent:**	Not Specified
Σ Original Estimate:	6 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-76	Grafické prvky na nastavenie vlastnos...	Sub-task	Done	Branislav Makan
	ES-77	Get a Set funkcie pre skriptá každej ...	Sub-task	Done	Marek Matula
	ES-78	Naviazanie grafických prvkov s Get a ...	Sub-task	Done	Branislav Makan
Sprint:	Sprint III - Cadillac				
Story Points:	5				

[ES-66] Vylepšenie čiar Created: 07/Nov/16 Updated: 20/Nov/16 Resolved: 20/Nov/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Martin Cvicela
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	30 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	2 days, 30 minutes	**Time Spent:**	Not Specified

Σ Original Estimate:	1 day, 2 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-67	Ťahanie čiar	Sub-task	Done	Martin Cvicela
	ES-70	Select čiar	Sub-task	Done	Martin Cvicela
	ES-69	Mazanie čiar	Sub-task	Done	Martin Cvicela
	ES-68	Zalomenie čiar	Sub-task	Done	Martin Cvicela
Sprint:	Sprint III - Cadillac				
Story Points:	5				

[ES-71] Funkčný prototyp Created: 07/Nov/16 Updated: 21/Nov/16 Resolved: 21/Nov/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Adam Blaško
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		

Original Estimate:	Not Specified

Sprint:	Sprint III - Cadillac
Story Points:	13

1.15.4 Šprint Dodge

[ES-16] Pauza a štart Created: 12/Oct/16 Updated: 05/Dec/16 Resolved: 05/Dec/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Tomas Liscak
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	2 hours		
Time Spent:	1 hour		
Original Estimate:	3 hours		

Sprint:	Sprint IV - Dodge
Story Points:	3

[ES-74] Meracie body v el. obvode Created: 07/Nov/16 Updated: 03/Dec/16 Resolved: 03/Dec/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Marek Matula
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	0 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	6 hours	**Time Spent:**	Not Specified
Σ Original Estimate:	6 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-87	Odber hodnôt zo súčiastky	Sub-task	Done	
	ES-88	Súčiastka ampérmeter	Sub-task	Done	
	ES-89	Súčiastka voltmeter	Sub-task	Done	
Sprint:	Sprint IV - Dodge				
Story Points:	3				

[ES-80] Upgrade Properties Windowu Created: 21/Nov/16 Updated: 01/Dec/16 Resolved: 01/Dec/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Branislav Makan
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	1 hour, 15 minutes	**Remaining Estimate:**	Not Specified
Σ Time Spent:	2 hours, 45 minutes	**Time Spent:**	Not Specified
Σ Original Estimate:	3 hours	**Original Estimate:**	Not Specified

Sub-Tasks:	**Key**	**Summary**	**Type**	**Status**	**Assignee**
	ES-90	Input validácia	Sub-task	Done	Branislav Makan
	ES-91	Nové grafické prefabs	Sub-task	Done	Branislav Makan
Sprint:	Sprint IV - Dodge				
Story Points:	3				

[ES-79] Alpha Testing I Created: 21/Nov/16 Updated: 11/Dec/16 Resolved: 11/Dec/16	
Status:	Done
Project:	EduSim
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Story	**Priority:**	Medium
Reporter:	Branislav Makan	**Assignee:**	Martin Cvicela
Resolution:	Done	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	0 minutes	**Remaining Estimate:**	0 minutes
Σ Time Spent:	7 hours, 26 minutes	**Time Spent:**	1 hour
Σ Original Estimate:	2 hours, 6 minutes	**Original Estimate:**	Not Specified

Sub-Tasks:

Key	Summary	Type	Status	Assignee
ES-81	Deselect	Sub-task	Done	Martin Cvicela
ES-82	Padá simulácia, keď nič nie je zapojené	Sub-task	Done	Tomas Liscak
ES-83	Delete na pripojenú súčiastku dá error	Sub-task	Done	Martin Cvicela
ES-84	Select súčiastky a čiary naraz	Sub-task	Done	Martin Cvicela
ES-85	Kreslenie čiar	Sub-task	Done	Martin Cvicela

		medzi uylom a súčiastkou			
	ES-86	Uzol v tollboxe sa dá selektovať	Sub-task	Done	Martin Cvicela
	ES-92	Rezistor a cievka sa nie vzdy draguje...	Sub-task	Done	Martin Cvicela
	ES-93	Simulacia pada pri vymazani a opatovn...	Sub-task	Done	Tomas Liscak
	ES-94	Connector ako komponent zle rescaluje	Sub-task	Done	Ivan Gulis
	ES-95	Niekedy pri dragovani necekuje koliziu	Sub-task	Done	Martin Cvicela
	ES-96	Zly checkollision suciastok	Sub-task	Done	Branislav Makan
	ES-98	Vytvaranie ciarotovanej suciastky ...	Sub-task	Done	Martin Cvicela
Sprint:	Sprint IV - Dodge, Sprint V - Eagle				
Story Points:	3				

2 Inžinierske dielo

2.1 Úvod k inžinierskemu dielu

V súčasnosti väčšina prednášok je vo forme prezentácií. Typicky, študenti sedia v aule a počúvajú prednášajúceho. Takáto forma prednášok zvykne byť nezaujímavá a vyčerpávajúca, keďže študent pri tom nič nerobí. Prezentácie teda nie sú najlepšou formou na udržanie pozornosti študenta. Prednášky sa stávajú trápnosťou a študenti sa im začnú vyhýbať. Je známe, že najlepší spôsob učenia, je skúšať si veci.

Keďže sa hry celkovo stali súčasťou moderného života mnohých ľudí, v našom projekte sa snažíme spojiť tie dve činnosti dokopy. Vytváraný nástroj by mal mať formu hry a minimálne interaktívnym spôsobom obohatiť proces výuky pre študentov vysokých škôl, ale aj mladších žiakov. Tento nástroj bude zároveň slúžiť aj ako pomôcka na vytváranie učebných materiálov pre pedagógov.

Tento projekt vznikol v spolupráci s firmou Atos. Hlavná idea je vytvoriť nástroj, pomocou ktorého sa budú vytvárať interaktívne simulácie, pomocou ktorých sa budú môsť testovať aj vedomosti žiakov a študentov. Vytvorené simulácie sa budú exportovať do HTML5 webových stránok, čo umožní jednoduchý prístup každému , keďže stačí mať prehliadač s pripojením na internet. Testovacie moduly budú prepojené s existujúcimi testovacími systémami.

Softvér je implementovaný v Unity game engine. Bude podporovať viaceré edukačné moduly, ktoré sa budú ľahko vymieňať. V prvej iterácii, teda v rámci predmetu tímový projekt, sa vytvorí jadro nástroja a modul pre elektrické obvody.

2.2 Globálne ciele

Cieľom nášho projektu je vytvorenie autoringového nástroja, ktorý bude slúžiť ako nástroj na zlepšenie výučby rôznych predmetov. Ako taký musí byť nástroj všeobecný a schopný simulovať rôzne domény akými sú napríklad elektrotechnika, fyzika, chémia a ďalšie. Z toho vyplýva, že nástroj musí byť modulárny a rozšíriteľný. Na základe preferencií zákazníka - product ownera sa ako prvá doména vyvinie elektrotechnika a ďalšie domény sa podľa preferencií vyvinú dodatočne. Tieto autoringové simulácie musí byť možné z nástroja vyexportovať vo forme HTML stránky. Samotný nástroj bude postavený na platforme Unity.

2.2.1 Zimný semester

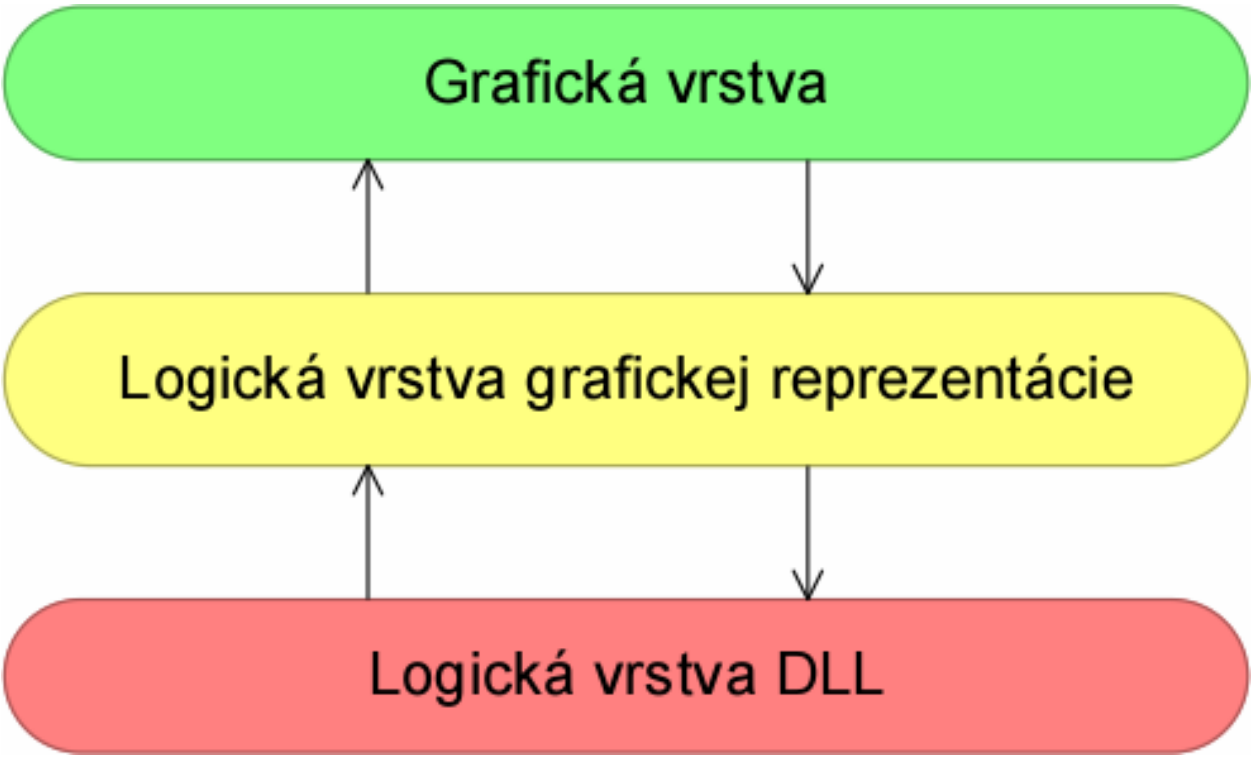
Základným cieľom nášho projektu pre zimný semester je vytvorenie funkčného prototypu výsledného produktu - MVP. Tento funkčný prototyp musí byť použiteľný produkt, ktorého využitie v praxi má zmysel aspoň do istej miery. Z toho vyplýva, že náš funkčný portotyp musí byť schopný modelovať a simulovať doménu elektrotechniky. Okrem funkčnej simulácie musí produkt samozrejme poskytovať aj podpornú funkcionalitu, bez ktorej by simulácia sama o sebe nemala žiaden význam. Takouto podpornou funkcionalitou sa myslí používateľské rozhranie, funkčná pracovná plocha na modelovanie elektrotechnických obvodov, na ktorých sa simulácia bude vykonávať, grafická reprezentácia elektrotechnických súčiastok a možnosť nastavovania parametrov týchto súčiastok.

2.2.2 Letný semester

Ciele pre letný semester budú priebežne formulované product ownerom. Medzi všeobecné ciele však patrí integrácia so vzdelávacími systémami (napr. moodle), možnosť testovania študentov v nástroji, prípadne pridanie ďalších domén pre simulácie - napr. fyzika, dynamika, chémia. V rámci letného semestra sa taktiež súčastníme súťaže TP cup, tým pádom ďalším cieľom je príprava na túto súťaž.

2.3 Celkový pohľad na systém

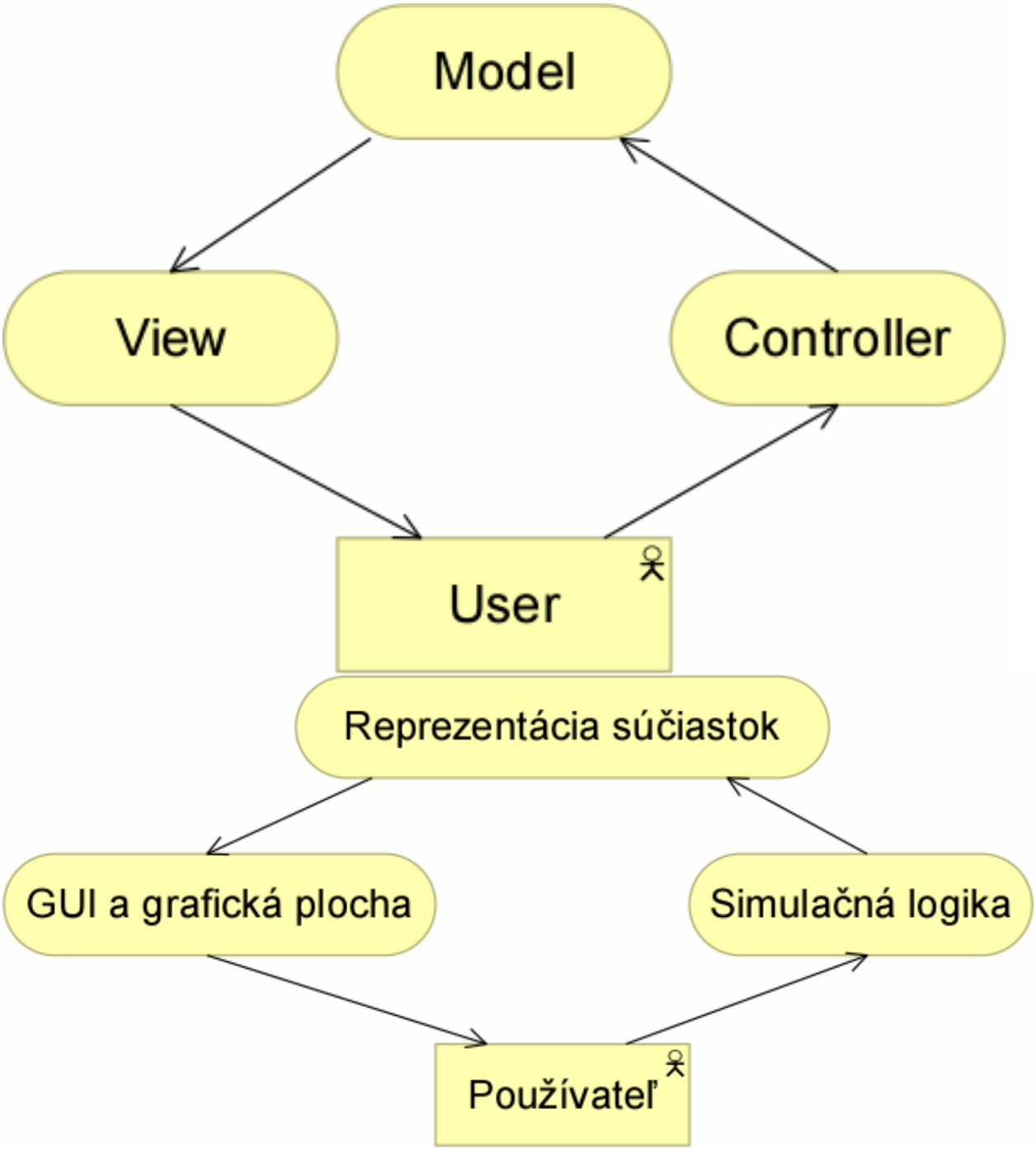
2.3.1 Dátový model - vrstvová štruktúra



V našom systéme sme identifikovali vrstvovú architektúru.

1. **Grafická vrstva** - Komunikuje s Logickou vrstvou grafickej reprezentácie, ktorej poskytuje súbor vykreslených súčiastok na pracovnej ploche, ako aj informácie o ich umiestnení, logickom prepojení a používateľom nastavené atribúty.
2. **Logická vrstva grafickej reprezentácie** - Komunikuje s grafickou vrstvou a vytvára programovú reprezentáciu objektov pracovnej plochy a simulácie ako celku. Spúšťa samotnú simuláciu pomocou výpočtových funkcii Logickej vrstvy DLL. Pre použitie týchto funkcii tiež mapuje všetky získané grafické súčiastky na ich objektové reprezentácie v DLL.
3. **Logická vrstva DLL** - Poskytuje algoritmy na výpočty požadovaných hodnôt. Informácie posiela do Logickej vrstvy grafickej reprezentácie, ktorá aktualizuje programovú reprezentáciu a následne všetko zobrazí v Grafickej vrstve.

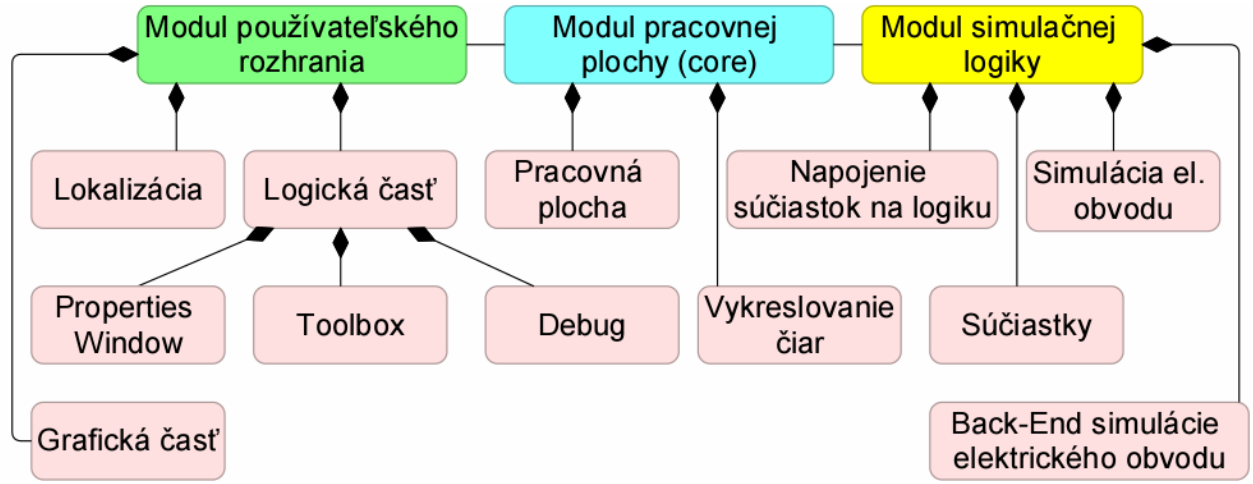
2.3.2 Model architektúry - MVC



Ako architektonický model používame architektúru MVC (Model-view-controller).

1. **Reprezentácia súčiastok** - Komponent Model, ktorý uchováva všetky informácie o súčiastkach a objektoch, s ktorými simulácia pracuje. Poskytuje pre View informácie, ktoré je potrebné zobraziť.
2. **GUI a grafické plocha** - Komponent View, ktorý používateľovi zobrazuje informácie.
3. **Simulačná logika** - Komponent Controller, ktorý riadi celú simuláciu a manipuláciu so súčiastkami. Poskytuje používateľovi funkcie nad súčiastkami.

2.3.3 Diagram modulov systému



Náš vytváraný systém v súčasnosti obsahuje 3 väčšie moduly, ktoré sa delia na menšie časti:

1. **Modul používateľského rozhrania** - V tomto module sa vyvíja všetko ohľadom GUI. Od tvorby hlavného rozhrania, cez lokalizáciu (slovenčina-angličtina), po implementáciu jednotlivých komponentov (Properties, Toolbox, Debug...).
2. **Modul pracovnej plochy (core)** - V tomto module sa vyvíja všetko ohľadom pracovnej plochy. To zahŕňa funkcie na prácu s objektami a funkcie na vykresľovanie a lámanie čiar na spájanie objektov.
3. **Modul simulačnej logiky** - V tomto module sa vyvíja všetko ohľadom simulačnej logiky. Obsahuje grafový algoritmus na odstraňovanie uzlov, knižnicu na výpočtové algoritmy, mapovanie a programovú reprezentáciu súčiastok elektrického obvodu.

2.3.4 Diagramy tried

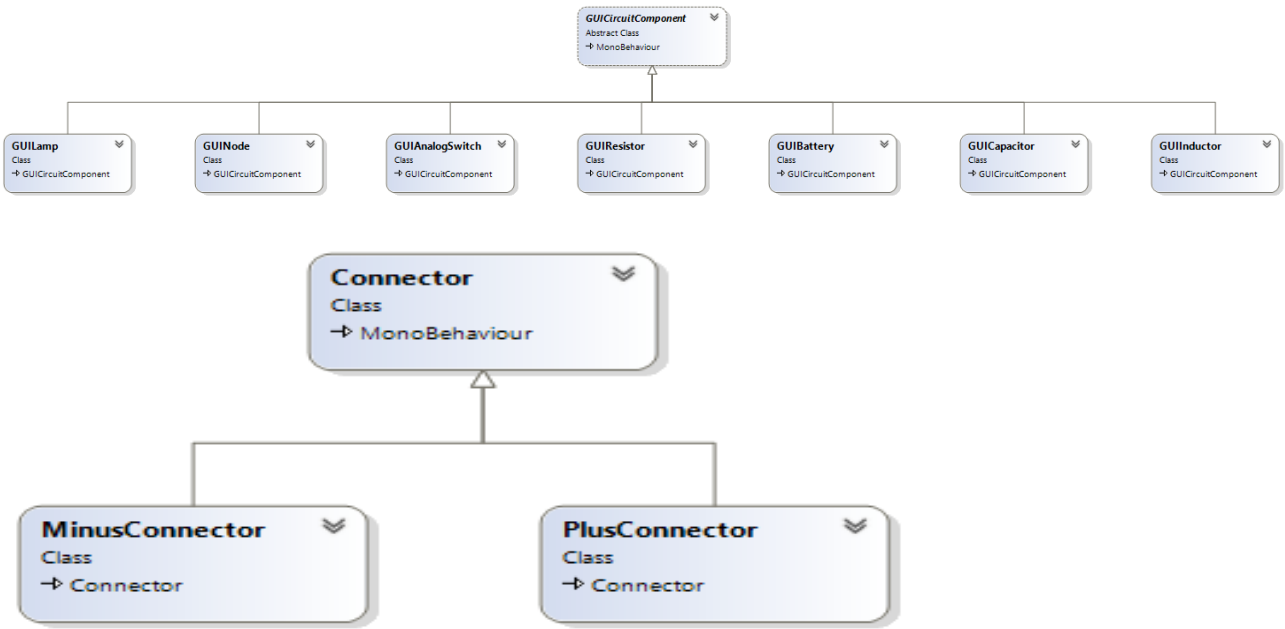


Diagram tried sa skladá z dvoch častí:

1. Hierarchia **Komponentov** - obsahuje programovú reprezentáciu grafických súčiastok elektrického obvodu.
2. Hierarchia **Connectorov** - obsahuje reprezentáciu konektorov, cez ktoré sa súčiastky spájajú vodičmi (existuje konektor kladný a záporný).

3 Opis modulov systému

3.1 *Používateľské rozhranie*

3.1.1 GUI - grafická časť [Audi]

Analýza

GUI predstavuje rozhranie, pomocou ktorého používateľ komunikuje so systémom.

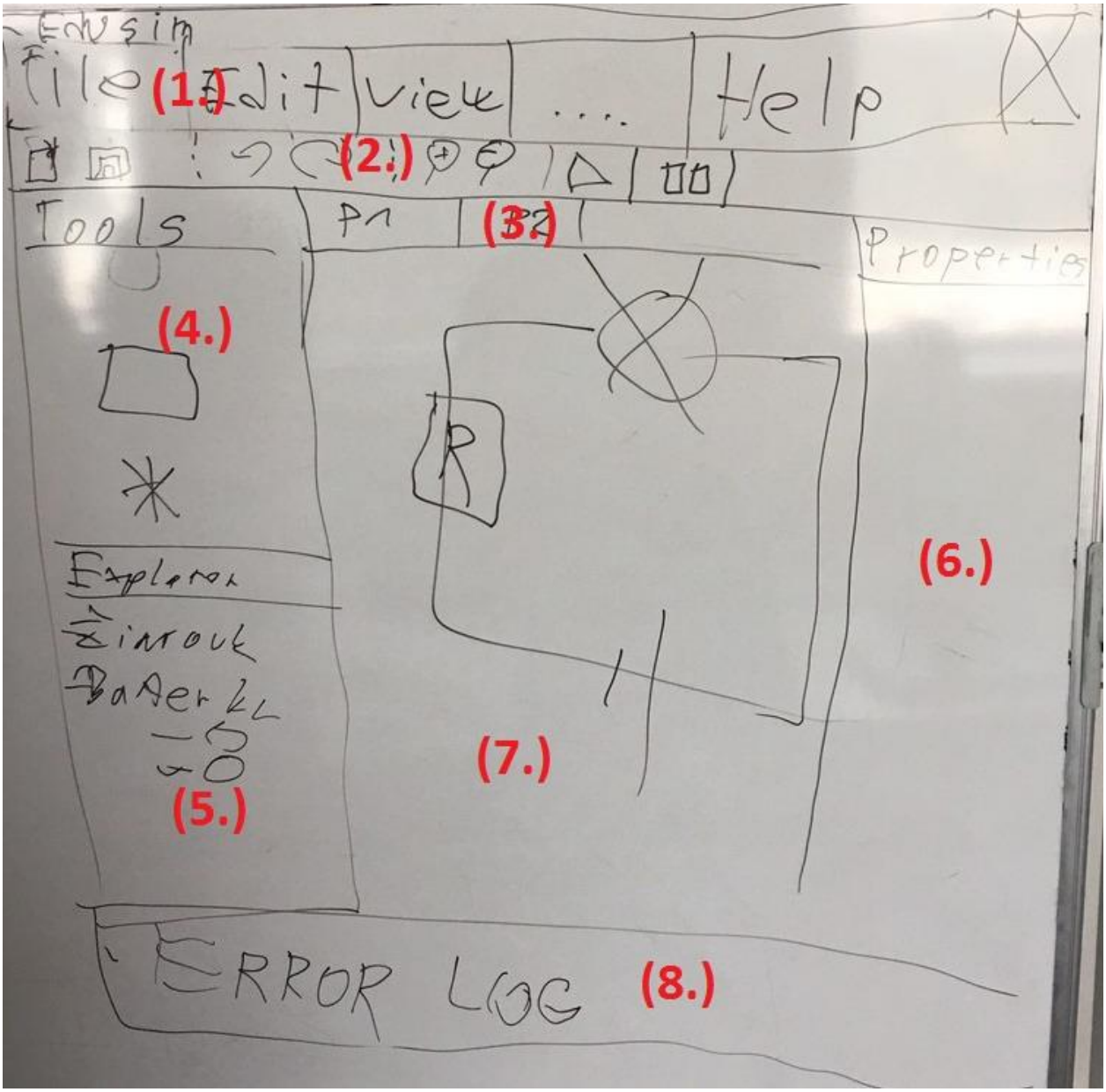
Malo by spĺňať určité kritériá:

1. **intuitívnosť** - nápisy musia byť jednoznačné
2. **výstižnosť** - ikonky tlačidiel musia vystihovať ich funkcionalitu
3. **familiárnosť** - kompozícia a vzhľad elementov by mal byť čo najviac podobný iným používaným aplikáciam

Používateľské rozhranie samotného implementačného prostredia Unity je možné do určitej miery zobrať ako vzor pre naše GUI.

Každá funkcia systému by mala mať aj svoj GUI komponent.

Návrh

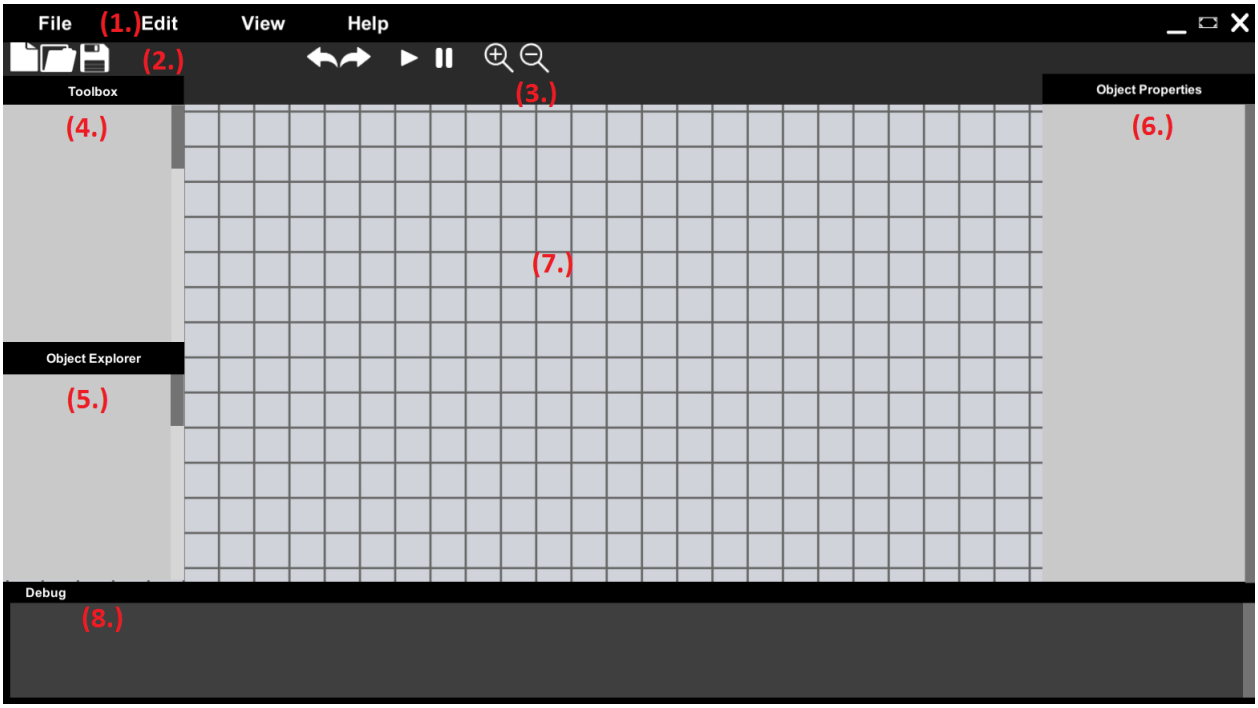


Jednoduchý návrh, ktorý sme načrtli na tabuľu, sa skladá z 8 väčších komponentov, ktoré sa ďalej členia na menšie objekty.

1. **Main menu** - panel by mal obsahovať základné textové menu s výberom možností po rozkliknutí
2. **Toolbar** - panel by mal obsahovať základné tlačidlá pre riadenie simulácie, grafickej plochy a projektu ako takého
3. **Tabs** - panel by mal obsahovať taby v prípade, že bude v jednom projekte existovať viac pohľadov
4. **Toolbox** - panel by mal obsahovať hlavičku s názvom a priestor pre všetky objekty, ktoré je možné pridávať do projektu na pracovnú plochu
5. **Object Explorer** - panel by mal obsahovať hlavičku s názvom a priestor pre objekty vytvoreného projektu

6. **Object Properties** - panel by mal obsahovať hlavičku s názvom a priestor pre atribúty označeného objektu z pracovnej plochy
7. **Pracovná plocha** - plocha s aktívnymi objektami projektu, s ktorými je možné manipulovať a simulovať ich správanie
8. **Error Log** - panel s priestorom na výpisy

Implementácia



Každý komponent z návrhu má svoju grafickú implementáciu.

1. **Main menu** - obsahuje 7 tlačidiel:
    1. 4 tlačidlá na ľavej strane, ktoré po rozkliknutí otvoria svoje menšie menu textových tlačidiel
    2. 3 tlačidlá na pravej strane, ktoré reprezentujú bežné "Minimize", "Maximize" a "Close" funkcie
2. **Toolbar** - obsahuje 9 tlačidiel s ikonkami:
    1. prvé 3 reprezentujú funkcie pre ovládanie projektu ako celku, "New", "Open" a "Save"
    2. druhé 2 reprezentujú funkcie "Undo" a "Redo"
    3. tretie 3 reprezentujú funkcie "Play" a "Pause" pre samotnú simuláciu
    4. posledné 2 reprezentujú funkcie pre priblíženie a oddialenie pracovnej plochy
3. **Tabs** - neobsahuje žiadne prvky, taby budú generované scriptom
4. **Toolbox** - obsahuje hlavičku ako Text Field a scrollbar na posúvanie obsahu
5. **Object Explorer** - obsahuje hlavičku ako Text Field a scrollbar na posúvanie obsahu
6. **Object Properties** - obsahuje hlavičku ako Text Field a scrollbar na posúvanie obsahu
7. **Pracovná plocha** - tento komponent bol implementovaný zvlášť Pracovná plocha [Audi, Bentley]
8. **Debug** - premenovaný z "Error Log", obsahuje panel s hlavičkou ako Text Field, scrollbar na posúvanie výpisov a Text Field na výpisy

Testovanie

Keďže v tejto časti nebola implementovaná žiadna funkcionalita, testovanie nie je potrebné. Každý komponent má svoju grafickú reprezentáciu je postačujúci test.

3.1.2 GUI - logická časť I [Bentley]

Analýza

GUI - logická časť 1 sa skladá z implementácie dvoch hlavných komponentov - Toolboxu a Debugu.

Toolbox predstavuje komponent, v ktorom sú zhromaždené všetky použiteľné komponenty domény projektu. Tieto objekty používateľ presúva do pracovnej plochy, kde s nimi ďalej pracuje a tvorí podklad pre simuláciu.

Komponenty v Toolboxe nie je možné spájať, slúžia iba ako ikony. Tieto ikony potom generujú inštancie objektu, ktorý predstavujú.

Unity nepodporuje ako GUI elementy samotné objekty scény, takže Toolbox musí byť naplnený objektami s obrázkom ako atribútom.

Pohybovanie a klonovanie objektov už bolo implementované v Pracovná plocha [Audi, Bentley].

Hlavným problémom bude schovávanie objektov mimo Toolbox panelu, interagovať bude možné len s objektami v rámci Toolbox časti.

Debug panel predstavuje komponent, ktorý slúži na výpisy o udalostiach v simulácii.

Je nutné implementovať tzv. listener, ktorý sa bude starať o pridávanie správ do Text Fieldu a scrollovanie na aktuálnu správu.

Návrh

Oba panely budú používať svoj **scrollbar**, ktorý bude pripevnený k väčšiemu panelu na pozadí. Tento panel bude viditeľný len v rámci nehybnej vyrezanej časti, a bude sa posúvať vertikálne.

Do **Toolbox** panelu budú umiestnené jednotlivé objekty domény, ktoré budú mať nastavený tvoj tag označujúci predmet Toolboxu.

Každý objekt Toolboxu bude mať pridaný Image zhodný s textúrou, aby bol používateľom viditeľný. Pri vytiahnutí objektu z panelu sa mu zmení tag, takže sa bude správať ako objekt simulácie.

Do **Debug** panelu bude umiestnený jeden Text Field, kam sa budú scriptom posielať výpisy. Pre simuláciu výpisov dostane každý objekt domény script na posielanie správ.

Funkcie Debugu však budú môcť používať aj priamo súčiastky, bez nutnosti vlastnenia scriptu na posielanie správ.

Implementácia

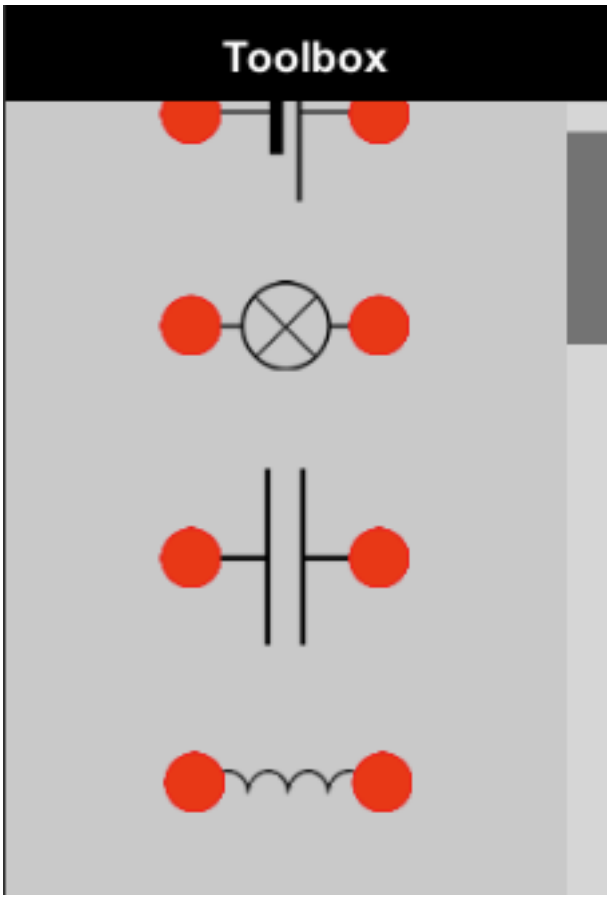
Toolbox je implementovaný pomocou komponentu Scroll Rect, ktorý obsahuje Viewport ,Masku a scrollbar. Marka zakrýva všetko mimo Viewportu.

Viewport obsahuje panel ako kontajner, v ktorom sú umiestnené jednotlivé objekty domény. Pri pohybovaní scrollbaru sa tento panel pohybuje.

Všetky objekty v kontajneri majú tag ToolboxItemActive, čo umožnuje zakázať pohyb, a umožňuje vytvárať jeho klony (inštancie). Zároveň majú tieto objekty vypnutý Sprite Renderer, a zapnutý Image, takže sú vidieť.

Pri vytiahnutí objektu z Toolboxu sa scriptom aktivuje Sprite Renderer a zmení sa tag na ActiveItem, takže už nie je možné objekt ďalej klonovať, a je možné s ním interagovať funkciami pracovnej plochy.

Zabránenie interakcie s objektami mimo Viewportu je zabezpečené pomocou komponentu na kamere - 2D Raycaster. Tento komponent povoluje interakciu len s objektami viditeľnými kamerou.



Debug tiež obsahuje Scroll Rect, ale v kontajneri je len jediný Text Field. Scrollbarom sa potom posúvajú riadky textu.

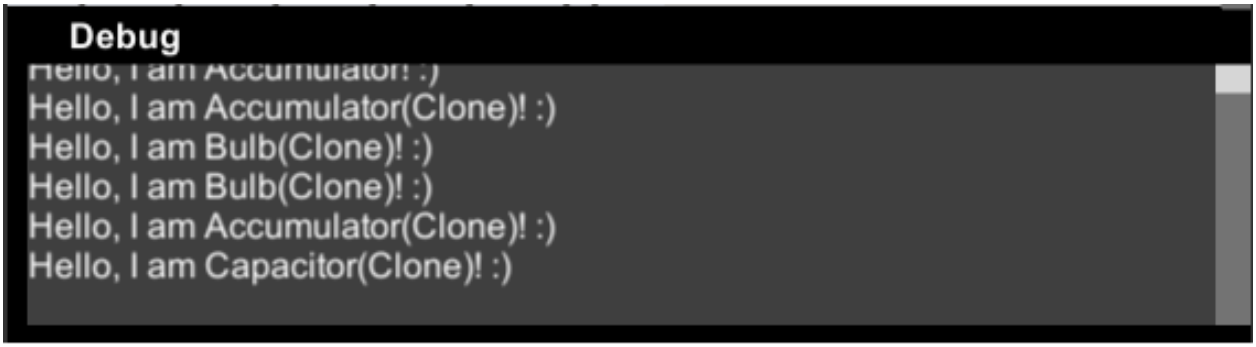
File: *Assets/GenericScripts/DebugDisaplay.cs*

Predpoklady:

- skript je v Canvas/DebugPanel/DebugLog ako komponent.
- skript má priradený Text Field, do ktorého má vypisovať.

Funkcie:

- *Write(string arg0)* - pridá string v argumente do Text Fieldu a scrollne na aktuálny posledný výpis



Testovanie

Toolbox bol otestovaný pridaním skupiny objektov súčiastok do panelu, overením funkcie scrollovania a vyťahovaním na plochu. Samotné funkcie vyťahovania a klonovania boli implementované v Pracovná plocha [Audi, Bentley].

Ako testovanie **Debugu** bol vytvorený script "Whisp.cs", ktorý bol pridaný každej súčiastke ako komponent.

Pri vytiahnutí súčiastky na pracovnú plochu potom vypíše jednoduchú správu.

File: *Assets/GenericScripts/Whisp.cs*

Predpoklady:

- skript je v Canvas/Toolbox/Scroll View/Viewport/Container/"súčiastka" ako komponent.

Funkcie:

- *Say(string arg0)* - vypíše jednoduchú správu "Hello, I am X! ", kde X = meno objektu (ako je vidieť na obrázku vyššie v časti Implementácia).

3.1.3 Properties Window [Cadillac, Dodge]

Analýza

Properties Window prestavuje spoj UI elementov Unity, selektovania aktívnych komponentov a simulačnej logiky. Ideo je, že sa pri selektovaní aktívneho komponentu zobrazia jeho nastaviteľné aktribúty v Properties okne.

Unity natívne podporuje rôzne GUI elementy. Medzi nimi sú aj Text, ktorý je vhodný na názvy, InputField, do ktorého vieme zapisovať hodnoty, Slider, vhodný pre ohraničenie číselných hodnôt a Toggle, vhodný na booleovské hodnoty.

Samotnú selekciu už máme naimplementovanú.Stačí ju vhodne prepojiť s nastaveniami okna.

Zo simulačnej logiky potrebujeme get a set funkcie, ktorými budeme získavať a nastavovať atribúty týchto komponentov.

Hlavným problémom sa javí dynamickosť a generickosť Properties okna. Potrebujeme ho vytvárať dynamicky tak, aby komponenty neboli kvantitatívne obmedzené - treba vytvoriť pár Label, user input field pre každý nastaviteľný atribút komponentu. Treba mať na mysli aj generickosť, aby sa dalo ľahko pridať property pre nejaký iný komponent v budúcnosti.

Návrh

Ku každému atribútu selektovaného aktívneho komponentu je sa vytvára jeden ObjectProperty. ObjectProperty pozostáva z jedného Text elementu (ObjectPropertyLabel) a jedného user input elementu (InputField, Slider a InputField alebo Toggle). V ObjectPropertyLabel sa nachádza názov atribútu. V user input je aktuálna hodnota toho atribútu.

Pri selektovaní aktívneho komponentu nastanú dve veci:

- okno s vlastnosťami sa vyprázdni;
- popní sa údajmi selektovaného komponentu.

Na zobrazenie atribútov aktívneho komponentu sa volá jeho metóda getProperties(). Využíva polymorfizmus - metóda getProperties() je definovaná v spoločnej nadtriede Component2, od ktorej všetky elektrické komponenty dedia. V tejto metóde, každá súčiastka volá metódy skriptu EditObjectProperties.cs (Clear a rôzne Add) a zobrazuje svoje atribúty v okne.

Pri deselekcii sa okno vyprázdni (Clear).

Okno s vlastnosťami je editovateľné. InputField, Slider a Toggle sa môžu editovať, čím sa nastavujú atribúty aktívneho komponentu. Pri vytvorení každého user input fieldu-u je k nemu pripojená metóda Set(), ktorá čaká na vhodný event.

Implementácia

File: *Assets/GenericScripts/EditObjectProperties.cs*

Predpoklady:

- skript v Canvas/ObjectPropertyPanel/ScrollView/ViewPort/PropertiesWindowContainer ako komponent.

Funkcie:

- *Clear()* - vymaže všetky položky v Properties okne.
- AddNumeric(string resourceKey, string value, string validationType, Action<double> set, bool useSlider, float min = 0, float max = 100)
  - resourceKey - kľúč z XML resource file (pozri časť k Lokalizácii);
  - value - hodnota atribútu;
  - validationType - typ číselného atribútu;
  - set - Setter funkcia na nastavenie atribútu;
  - useSlider - použiť slider?;
  - min - nepovinný argument, minimálna hodnota atribútu;
  - max - nepovinný argument, maximálna hodnota atribútu.
- AddBoolean(string resourceKey, string value, Action<bool> set)
  - resourceKey - kľúč z XML resource file (pozri časť k Lokalizácii);
  - value - hodnota atribútu;

    - set - Setter funkcia na nastavenie atribútu.
- AddResult(string resourceKey, string value, string unit = "")
    - resourceKey - kľúč z XML resource file (pozri časť k Lokalizácii);
    - value - hodnota atribútu;
    - unit - nepovinný argument, jednotka (môže byť UTF kód na značku jednotky).

3.1.4 Lokalizácia - resource files [Bentley]

Analýza

Unity engine ako framework natívne nepodporuje žiadu formu lokalizácie textov, preto je potrebné vytvoriť si vlastný spôsob lokalizácie textov v aplikácií.

Návrh

Základom lokalizácie je súbor alebo množina súborov, ktoré sú nositeľmi lokalizovaných textov. Pre účely nášho projektu sme zvolili XML súbor, ktorého elementami sú konkrétne podporované jazyky. Tento XML súbor sa používa ako zoznam kľúčov a hodnôt pre potrebné elementy používateľského rozhrania.

Implementácia

Lokalizácia je implementovaná pomocou skriptu Localization.cs a triedy ResourceReader.cs. Lokalizované texty sa nachádzajú v súbore Resources.xml.

Localization

Pre správnu funkcionalitu lokalizácie je potrebné priradiť skript *Localization* spoločnému rodičovkému elementu všetkých textov, ktoré je potrebné lokalizovať. Pre zmenu používaného jazyka je potrebné volať metódu ChangeLanguage s argumentom jazyka, ktorý sa má použiť. Na definovanie jazykov sa používa SystemLanguage Enumaration, ktorý je definovaný v Unity. Pri prvom spustení sa nastavuje jazyk na jazyk systému.

ResourceReader

Slúži ako pomocná trieda pre parsovanie XML súboru, ktorý obsahuje lokalizovaný text. V prípade, že sa požaduje čítanie jazyka, ktorý nie je obsiahnutý v XML súbore, použije sa default jazyk - anglický.

Resources

Koreňovým elementom XMl súboru je *Languages*, ktorý obsahuje zoznam podporovaných jazykov prostredníctvom svojich detí. Elementy konkrétnych jazykov (napr. *Slovak*) obsahujú zoznam lokalizovaných textových reťazcov, kde kľúčom je atribút *name* a obsah elementu je nositeľom lokalizovaného textu. Ako kľúč pre mapovanie textov na Unity elementy sa používa názov unity elementov.

3.1.5 Kompletizácia grafiky I [Eagle]

Analýza

Kompletizácia grafiky 1. časť sa skladá z 3 pod-častí:

1. Pop-up okná pre tlačidlá z hlavného menu
2. Interaktivita tlačidiel hlavného menu
3. Interaktivita tlačidiel toolbaru

Okrem buttonov na tvorbu pop-upov musia byť interaktívne aj tlačidlá pre zobrazovanie/skrývanie bočných panelov Toolbox, Debug, Properties a Object Explorer. Tlačidlá Play a Pause v menu Edit musia kopírovať funkcionalitu tlačidiel Play a Pause z toolbaru - viditeľne označené, kedy sú stlačené. Tlačidlá Otvoriť, Uložiť a Uložiť ako musia interagovať rovnako ako tlačidlá z toolbaru - otvárať windows explorer.

Tlačidlá na otáčanie a mazanie súčiastok by sa mali zobrazovať len ak je nejaká súčiastka označená.

Návrh

Zoznam pop-up panelov, budú otvárané z hlavného menu tlačidlami rovnakých názvov:

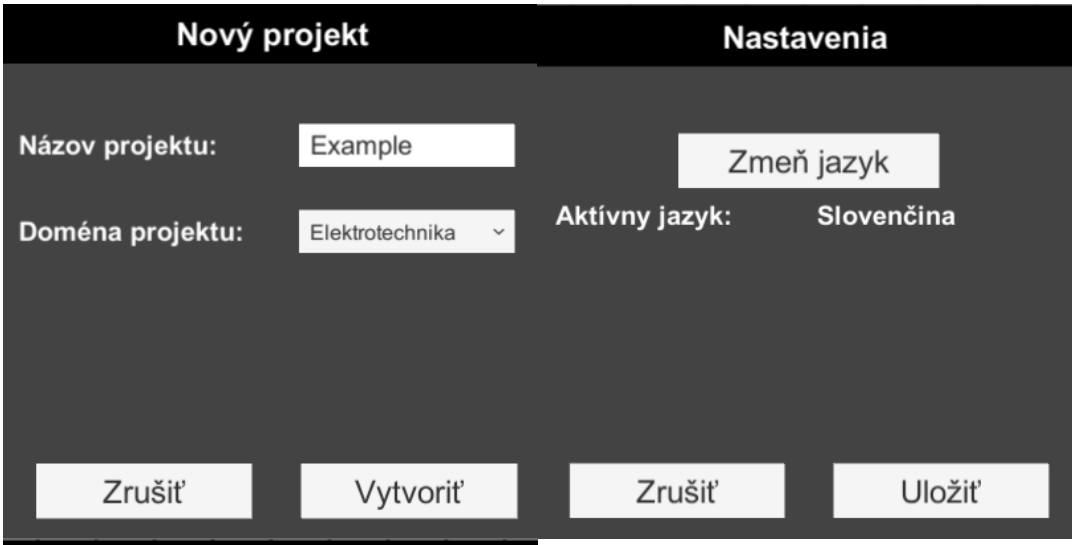
- Nový projekt - panel na tvorbu nového projektu
- Nastavenia - nastavenia programu
- O programe - panel s opisom programu
- O verzii - panel s vypísanou verziou programu
- Nahlásiť chybu - panel s možnosťou vyplnenia informácii o chybe
- Kontakt - panel s kontaktnými informáciami na náš tím

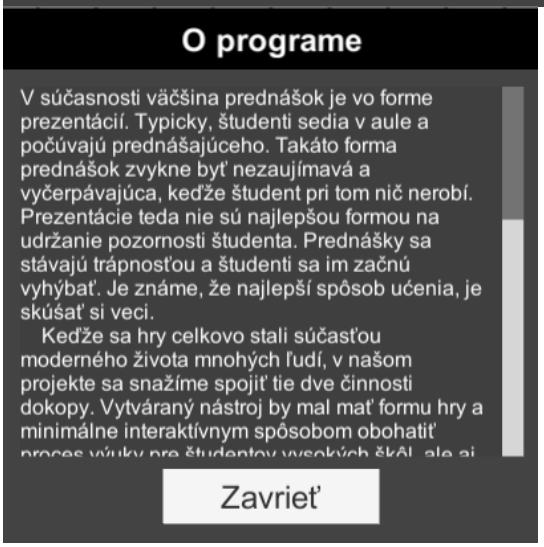
Zoznam tlačidiel hlavného menu, ktoré budú interaktívne:

- Nový projekt - otvorí okno Nový projekt
- Otvoriť, Uložiť, Uložiť ako - otvoria windows explorer
- Spustiť, Pozastaviť - spustia alebo pozastavia simuláciu
- Buttony View menu - budú ukazovať a skrývať bočné panely

Zoznam tlačidiel toolbaru, ktoré budú interaktívne:

- Nový projekt, Otvoriť, Uložiť, Uložiť ako, Play, Pause, Priblíženie, Oddialenie, Otočenie doľava, Otočenie doprava, Zmazanie


Nový projekt
Názov projektu:
Example
Doména projektu:
Elektrotechnika
Zrušiť
Vytvoriť
Nastavenia
Zmeň jazyk
Aktívny jazyk:
Slovenčina
Zrušiť
Uložiť


O programe
V súčasnosti väčšina prednášok je vo forme prezentácií. Typicky, študenti sedia v aule a počúvajú prednášajúceho. Takáto forma prednášok zvykne byť nezaujímavá a vyčerpávajúca, keďže študent pri tom nič nerobí. Prezentácie teda nie sú najlepšou formou na udržanie pozornosti študenta. Prednášky sa stávajú trápnosťou a študenti sa im začnú vyhýbať. Je známe, že najlepší spôsob učenia, je skúšať si veci.
Keďže sa hry celkovo stali súčasťou moderného života mnohých ľudí, v našom projekte sa snažíme spojiť tie dve činnosti dokopy. Vytváraný nástroj by mal mať formu hry a minimálne interaktívnym spôsobom obohatiť
Zavrieť

Implementácia pop-up panelov, každý panel bude obsahovať 1-2 tlačidlá - na zrušenie a niektoré aj na pokračovanie:

- Nový projekt - 2 informatívne labely, 1 inputField na vyplnenie mena projektu a 1 dropdown na výber domény projektu
- Nastavenia - 1 button na točenie slovenského a anglického jazyka a 2 informatívne labely s aktuálnym jazykom
- O programe - 1 textové pole s krátkym textom
- O verzii - 2 informatívne labely s verziou programu
- Nahlásiť chybu - 1 informatívny label a inputField na písanie správy
- Kontakt - 2 informatívne labely s tímovým e-mailom

Implementácia tlačidiel toolbaru, ktoré budú interaktívne:

- Nový projekt - otvorí okno Nový projekt

- Otvoriť, Uložiť, Uložiť ako - otvoria windows explorer
- Spustiť - spustí simuláciu a zafarbí sa (odfarbí Pause button)
- Pozastaviť - zastaví simuláciu a zafarbí sa (odfarbí Play button)
- Priblíženie - priblíži grafickú plochu a pripočíta percentá do textového poľa
- Oddialenie - oddiali grafickú plochu a odráta percentá z textového poľa
- Otočenie doľava - označenú súčiastku otočí o 90 stupňov v protismere hodín
- Otočenie doprava - zmaže označenú súčiastku o 90 stupňov v smere hodín
- Zmazanie - zmaže ozačenú súčiastku

Funkcionalita bola doimplementovaná do existujúcich scriptov z grafickej plochy a kamery, a namapovaná na tlačidlá.

Tlačidlá Otočenie doľava, Otočenie doprava a Zmazanie sa aktivujú pri označení súčiastky, a deaktivujú pri jej odznačení.

Implementácia tlačidiel hlavného menu, ktoré budú interaktívne:

- Tlačidlá v menu Zobrazenie skrývajú alebo ukazujú bočné panely. Skladajú sa z textu a checkboxu, ktorý je zaškrtnutý a tlačidlo je stlačené, ak je daný panel zobrazený.
- Tlačidlá Spustiť a Pozastaviť v menu Upraviť kopírujú funkcionalitu tlačidiel v toolbare - značenie je však rovnaké ako v menu Zobrazenie.

File: *Assets/Scripts/Menu buttons/MainMenuButtons.cs*

Predpoklady:

- Skript je v _MainMenuManager ako komponent.
- Objekt _MainMenuManager je priradený do OnClick() v tlačidlách, ktoré používajú metódy scriptu.

Funkcie:

- *PlayPauseButton(string action)* - riadi a označuje tlačidlá Play/Pause
- *Show"X"(GameObject guiComponent)* kde X = {Toolbox, Properties, ObjectExplorer, Debug} - zobrazia alebo skryjú bočný panel
- *OpenProject()* - otvorí windows explorer
- *SaveProject()* - otvorí windows explorer
- *SaveAsProject()* - otvorí windows explorer
- *Show"X"PanelMenu(GameObject guiComponent)* kde X = {File, Edit, View, Help} - zobrazia alebo skryjú dropdowny hlavného menu
- *Show"X"Canvas(GameObject guiComponent)* kde X = {NewProject, Settings, AboutProject, ReleaseNotes, ContactInfo, ReportBug} - zobrazia alebo skryjú pop-upy tlačidiel hlavného menu

Testovanie

Na overenie funkcionality bolo použité jednoduché testovanie, pri ktorom boli preklikané všetky tlačidlá hlavného menu a tak overená ich funkčnosť, vrátane tlačidiel Spustiť a Pozastaviť.

Pre overenie tlačidiel toolbaru bol vytvorený jednoduchý dvojsúčiastkový obvod, kto ktorom bola otestovaná funkcionalita tlačidiel Otočenie doľava, Otočenie doprava, Zmazanie.

Funkcionalita tlačidiel Priblíženie a Oddialenie bola rovnako otestovaná pri predošlom teste.

3.1.6 Pracovná plocha

Dôležitou súčasťou každého softvérového nástroja je pracovná plocha. Je to grafická časť nástroja, kde sa predpripravené komponenty medzisebou spájajú do logického celku, nastavujú sa ich parametre a koná sa logika daného nástroja. Dôležité sú preto aj ovládacie prvky, ktoré sa v danom nástroji budú používať. Keďže ide o editovací nástroj, kde predpripravené koponenty spájame a nastavujeme, rozhodli sme sa, že jadro ovládacieho systému bude princíp drag and drop (ťahaj a uloľni). Okrem toho, dôležité je vybrať si konkrétny aktívny komponent, s ktorým chceme v danom okamihu niečo urobiť. Tomuto hovoríme selectovanie (označenie). Opakom je deselectovanie, kde selectovaný komponent uvoľníme. Ďalšou dôležitou funkciou je tvorba nových a mazanie existujúcich inštancií - create a delete. Potom sú tu funkcie na pohyb a rotáciu komponentov, ako aj rôzne ovládacie prvky pre kameru (zoom, pohyb). Na úzadie pracovnej plochy sme sa rozhodli dať mriežku (grid).

Všetky tieto komponenty sú bližšie opísané v nasledujúcich podčastiach.

3.1.7 Select [Audi]

Analýza

Select slúži na vyznačenie komponentu, nad ktorým sa vykonajú nejaké funkcie. V príbuzných softvérových nástrojoch sa takáto funkcia rieši stlačením ľavým tlačidlom myšky. Unity natívne podporuje niekoľko rôznych eventov, ktorými sa odchití kliknutie určitým tlačidlom myšky. Dva najpoužívanejšie riešenia sú funkcie OnMouseDown a OnPointerClick.

Návrh

Selektovanie sa bude konať stlačením ľavým tlačidlom myšky na aktívny komponent pracovnej plochy. Na tento účel sa použije funkcia OnPointerClick. V nej vieme určiť, ktoré tlačidlo myšky

pozorujeme a je odolnejšia na chyby (napr. pri OnMouseDown stlačením na UI sa event spustí aj na aktívny komponent v úzadí). Na zdôraznenie selekcie vykreslíme k selektovanému aktívnemu komponentu štvorec z trhaných čiar.

Implementácia

File: *Assets/Scripts/GenericScripts/SelectObject.cs*

Predpoklady:

- aktívny komponent má pripojený skript ako Unity komponent,
- aktívny komponent má 2D Box Collider komponent,
- aktívny komponent má nastavený odkaz na SelectionBox prefab v skripte,
- kamera má Physics 2D Raycaster,
- v scénke sa nachádza EventSystem.

Funkcia OnPointerClick() poslúcha na stlačenie myškou. Po stlačení myškou sa v prípade potreby najprv vykoná deselekcia a až potom selekcia. V jednom okamihu môže byť selektovaný iba jeden aktívny komponent. Po selektovaní sa aktívnemu komponentu zapne *SpriteRenderer* komponent, ktorý obsahuje obrázok selection boxu.

Selektovaný komponent je prístupný volaním *SelectObject.SelectedObject* atribútu, ktorý je public a static, z hociktorého iného skriptu.

3.1.8 Deselect [Bentley]

Analýza

Deselektovanie sa v príbuzných softvérovných nástrojoch koná stlačením myškou mimo aktívnych komponentov na pracovnej ploche.

Návrh

V úzadí pracovnej ploche bude background (jednofarebný obrázok). Keď naneho stlačíme myškou, znemená to, že sa pred ním nič nenachádza a môžeme deselektovať selektovaný obrázok.

Implementácia

File: *Assets/Scripts/GenericScripts/Deselect.cs*

Predpoklady:

- v úzadí existuje background - obrázok,
- background má 2D Box Collider komponent,
- skript na deselektovanie je pripojený k backgroundu ako komponent.

Funkcia OnPointerClick() poslúcha na stlačenie myškou. Po stlačení myšky sa vykoná deselekcia - *SelectObject.SelectedObject* sa nastaví na *null*.

3.1.9 Pohyb [Audi, Bentley]

Analýza

Pohyb elementov sa v príbuzných softvérových nástrojoch stotožňuje s princípom drag and drop. Na tieto účely Unity ponúka vhodné funkcie OnMouseDown a trio funkcií OnBeginDrag, OnDrag a OnEndDrag. Alternatívne sa pohyb elementov vykonáva vo vopred definovaných veľkostiach posunu po plochy. Unity natívne podporuje klávesové vstupy.

Návrh

Pohyb bude implementovaný v dvoch tvaroch. Pohyb pomocou myšky sa uskutoční princípom drag and drop. Implementovaný bude funkciami OnBeginDrag, OnDrag a OnEndDrag, ktoré nám umožňujú veľmi dobre rozdeliť logiku pohybu do troch častí: začiatok, priebeh a koniec. Druhý tvar posunu bude vo vopred definovaných veľkostiach, ktoré budú presne také ako je veľkosť mriežky. Aktívny element sa bude zdanlivo pohybovať po mriežky.

Implementácia

File: *Assets/Scripts/GenericScripts/Draggable.cs*

Predpoklady:

- aktívny komponent má pridaný skript ako komponent,
- aktívny komponent má 2D Box Collider,
- kamera má Physics 2D Raycaster,
- v scénke sa nachádza EventSystem.

Drag and Drop

Jadro logiky tvoria tri natívne Unity funkcie, ktoré odchytávajú eventy:

- OnBeginDrag
- OnDrag
- OnEndDrag

public void OnBeginDrag(PointerEventData eventData)

Funkcia zachytáva event začiatku ťahania myškou. V tejto funkcii sa koná inicializácia pohybu GameObjectu:

- začiatočná pozícia myšky pri stlačení na GameObject sa uloží do premenej *_mousePos*;
- začiatočná poloha GameObjectu sa uloží do premennej *_itemPos*.

public void OnDrag(PointerEventData eventData)

Táto funkcia vykonáva samotné ťahanie predmetu po pracovnej plochy:

- počíta sa *mouseDiff* - rozdiel medzi začiatočnou pozíciou myšky a aktuálnou;
- na základe mouseDiff sa vypočíta aktuálna pozícia GameObjectu.

public void OnEndDrag(PointerEventData eventData)

Funkcia ukončuje pohyb:

- posledná pozícia GameObjectu z funkcie OnDrag sa zaokrúhľuje na najblyžšiu 0.5 hodnotu:
    - prenásobí sa dvojkou;
    - zaokrúhli sa;
    - vydelí sa dvojkou;
- pozícia sa uloží ako finálna pre aktuálny pohyb GameObjectu.

Pohyb pomocou klávesnici

Predpoklady:

- selektovaný aktívny komponent.

Logika pohybu klávesnicou je implementovaná vo funkcii Update, kde sa pozoruje stlačenie klávesov *W, A, S, D*. Veľkosť posunu je predurčená na 0.5f s opozdením 0.25 sekúnd (definované v atribúte *_delay*). V každom frame sa volá funkcia *_decreaseDelay*, ktorá zníži tento delay za delta čas - čas potrebný na ukončenie posledného framu.

3.1.10 Kolízie [Bentley]

Analýza

Pri umiestňovaní elementov na pracovnú plochu chceme zabrániť umiestneniu jedného aktívneho komponentu na druhý. Unity podporuje rôzne Box Collideri, ktoré riešia kolízie, ale potrebujú k tomu simulácie gravitačného poľa. V našej aplikácie sa takéto simulácie nedejú a drag and drop fizyka v Unity chápe ako teleportovanie, čo narušuje jej simulácie.

Návrh

Aby sme sa vyhli simulovaniu fyzických javov, ktoré sa bežne používajú v Unity hrách, rozhodli sme sa jednoducho prepočítať koncové koordináty aktívneho elementu pri konci jeho pohybu. Ak nastane kolízia dvoch aktívnych elementov, ten ktorý sa posledný pohyboval bude posunutý.

Implementácia

File: *Assets/Scripts/GenericScripts/Draggable.cs*

Prepoklady:

- aktívny komponent má pridaný skript ako komponent,
- aktívny komponent má 2D Box Collider,
- kamera má Physics 2D Raycaster,
- v scénke sa nachádza EventSystem,
- aktívne komponenty sú otagované ako *ActiveItem*.

Po skončení pohybu sa spúšťa funkcia _checkCollision(). V nej sa prepočítajú koordináty a veľkoste všetkých aktívnych elementov. Ak nastane kolízia, posledný posúvajúci predmed bude posunutý horizontálne alebo vertikálne - záleží od samotnej kolízii. Kolízia sa overuje rekurzívne, čo znamená, že sa aj po posunutí znovu overí či neprišlo ku kolízii s ďalším aktívnym elementom.

3.1.11 Rotácia [Audi]

Analýza

Rotácia sa väčšinou koná pomocou myšky stlačením na roh elementu, ktorý chceme otočiť. Alternatívne riešenie je využietie klávesnice. Rotovať možeme zvyčajne iba jeden element a musí byť predtým selektovaný.

Návrh

Rotácia sa bude klávesnicovými skratkami *Q* (doľava) a *E* (doprava) nad vopred selektovaným aktívnym elementom.

Implementácia

File: *Assets/Scripts/GenericScripts/Rotate.cs*

Predpoklady:

- selektovaný aktívny komponent.

Vo funkcii Update sa pozoruje stlačenie klávesov Q a E na klávesnici. Q rotuje selektovaný komponent doľava. E rotuje selektovaný komponent doprava.

3.1.12 CameraZoom [Audi]

Analýza

Približovanie a vzďaľovanie kamery sa v príbuzných softvérových nástrojoch implementuje kolieskom myšky. Niekedy je potrebné stlačiť kláves CTRL (hlavne, keď nástroj podporuje skrolovanie myškou). Unity podporuje zmenu pozície kamery ako hociktorého iného GameObjectu. Alternatívne, podporuje zmenu ortografickej veľkosti, ktorou sa ovplyvní zoom.

Návrh

Na zoomovanie budeme používať koliesko myšky. Skrolovaním nahor sa priblíži obsah pracovnej ploche a skrolovaním nadol sa vzdiali. Napevno nastavím maximálne a minimálne zoomovanie. Je to preto, lebo nepodporujeme zmenu veľkosti komponentov.

Implementácia

File: *Assets/Scripts/GenericScripts/CamZoom.cs*

Predpoklady*:*

- kamera je ortografická,
- na akmeru sa pridá skript ako komponent.

Vo funkcii *Start()* sa inicializujú:

- rýchlosť približovania/vzdiaľovania,
- maximálne priblíženie,
- maximálne vzdiaľenie.

Približovanie sa koná točením kolieska myšky nahor. Vzdiaľovanie sa koná točením kolieska myšky nadol. Zoomovanie sa koná nastavením atribútu *orthographicSize* v kamere.

3.1.13 Pobyb kamery [Bentley]

Analýza

Pohyb po pracovnej ploche sa zvyčajne rieši slajdermi (scroll bars). Občas sa použije aj pristup, kde sa pohyb koná stlačením stredného tlačidla myšky a potom sa ťahá do želaného smeru pohybu. Unity natívne podporuje input z myšky a povoľuje nastavenie pozície kamery, čo nám dovoľuje nehýbať všetkým na ploche, ale iba kamerou.

Návrh

Pohyb bude implementovaný pomocou myšky. Pri stlačení stredného tlačidla a následného ťahania myškou na strany sa vykoná pohyb kamery, čím sa vytvorí ilúzia posunu celej plochy.

Implementácia

File: *Assets/Scripts/GenericScripts/CameraMovement.cs*

Predpoklady*:*

- existue background v úzadí - obrázok,
- k backgroundu je pripojený skript,
- k skriptu je nastavená referencia na kameru,
- background obsahuje 2D Box Collider,
- kamera obsahuje Physics 2D Raycaster.

Pohyb kamery sa koná stlačením stredného tlačidla myšky - kolieska, v dvoch fázach:

- pri stlačení sa uchovajú začiatočné pozície myšky a kamery;
- pri pohybe myškou sa tieto pozície menia.

3.1.14 Vytvorenie inštancií komponentu [Audi]

Analýza

Toolbox zvyčajne obsahuje preddefinované objekty, ktoré sa budú používať. Pomocou drag and drop princípu sa tieto objekty presunú na plochu čím sa vytvoria nové inštancia týchto objektov. Unity podporuje funkciu Instantiate, ktorá vytvorí klóna pôvodného objektu.

Návrh

Preddefinované objekty budú uložené v toolboxe. Pomocou drag and drop funkcie ich budeme prenášať na plochu, čím vytvoríme nové inštancie týchto objektov.

Implementácia

File: *Assets/Scripts/GenericScripts/Draggable.cs*

Predpoklady:

- komponent má pridaný skript ako komponent,
- komponent má 2D Box Collider,
- kamera má Physics 2D Raycaster,
- v scénke sa nachádza EventSystem,
- komponent je otagovaný ako *ToolboxItemActive*.

V kroku začiatku drag and drop, vo funkcii *OnBeginDrag()* sa vytvorí nová inštancia objektu, ktorá sa ďalej ťahá na plochu.

3.1.15 Mazanie inštancii komponentu [Bentley]

Analýza

Na mazanie elementov sa zvykne používať tlačidlo delete alebo pop-up okno v ktorom sa zvolí mazanie. V Unity existuje funkcia Destroy, ktorá zmaže danú inštanciu GameObjectu.

Návrh

Pomocou klávesu Del sa zmaže selektovaný aktívny komponent. Ak je komponent pripojený čiarou na iný komponent, musia sa vymazať aj tieto čiary a aktualizovať zoznamy pripojených konektorov v súčiastkach, ktoré boli s vymazávanou súčiastkou spojené.

Implementácia

File: *Assets/Scripts/GenericScripts/Destroy.cs*

Predpoklady:

- selektovaný aktívny komponent.

Po stlačení klávesu *Del sa* najskôr overí, či selektovaný komponent nie je čiarou pripojený k iným komponentom. Funkcia získa listy čiarou pripojených konektorov k danej súčiastke. Pre každý nájdený konektor sa následne vymaže zo zoznamu pripojených konektorov konektor zmazávanej súčiastky. Ďalej sa nájdu všetky čiary, ktoré vychádzali z konektorov vymazávanej súčiastky a zmažú sa. Nakoniec sa vymaže súčiastka.

3.1.16 Background [Bentley]

Analýza

Defaultne, Unity úzadie je modré. V Unity sa úzadie zvykne nastavovať ako obrázok. Alternatívou je prefarbiť úzadie v kamere (z modrého na niečo iné). Background zobrazený ako obrázok v úzadí nám však umožňuje zachytávať na ňom eventy.

Návrh

Umierstnenie jednofarebného obrázka do úzadia s box colliderom. Toto sa využije na zachytávanie eventov, napr. pre Deselect.

Implementácia

File: *Assets/Prefabs/Background.cs*

Prepoklady:

- prefab je umiestnený v scénke.

3.1.17 Grid [Audi, Bentley]

Analýza

Na úzadí pracovných plôch príbuzných softvérových nástrojov zvykne byť mriežka. Táto napomáha používateľovi pri umiestnení elementov. Unity umožňuje viacero prístupov na implementovanie mriežky, ani jeden však nie natívne. Pri vyhľadávaní sme sa stretli s viacerími riešeniami a viecerés vyskúšali.

Návrh

Mriežka je umiestnená ako textúra na štvorci. Táto textúra sa dynamicky vykreslí na základe parametrov.

Implementácia

File: *Assets/Scripts/GenericScripts/Grid.cs*

Prepoklady:

- prefab grid je v scénke,
- v prefabe je skript.

V scripte je možné nastaviť veľkosť mriežky, počet riadkov a stĺpcov. Mriežka je umiestnená v pozadí, aby neprekrývala žiadne komponenty na pracovnej plochy. Odporúčané je použiť dvojnásobok počtu riadkov a stĺpcov než je veľkosť mriežky - vtedy každý štvorček bude veľkosti 0.5f x 0.5f.

3.1.18 Skratky - hotkeys [Eagle]

Skratky v softvéri slúžia na urýchlenie častých akcií. Prestavujú pridanú hodnotu pre pokročilých používateľov, pre ktorých je jednoduchšie zapamatäť si zopár skratiek, ako často opakovať akcie, ktoré požadujú väčší počet akcií alebo používanie myši.

Úlohou tohoto modulu je vytvoriť centralizované miesto v kóde, kde budú skratky definované, a zároveň z tohoto miesta budú perzistovateľné a upravovateľné. Pre použiťie softvéru nestačí mať skratky pevno definované v kóde.

Analýza

Unity poskytuje triedu InpuManager, vďaka ktorej sa dajú definovať rôzne schémy používania pre rôzne vstupné periférie - rôzne joysticky, myš, klávesnica alebo dotyková obrazovka. Bohužiaľ, táto implementácia nie je dostatočná, nakoľko nepovoľuje meniť ovládaciu schému počas behu aplikácia. V iných ohľadoch je zas ťažkopádna, nakoľko v našom projekte nepotrebujeme podporu pre joysticky. Takisto, nakoľko podporuje joysticky, tak táto ovládacia schéma podporuje rôzne "analógové" hodnoty, respektíve využíva koncept rôznych osí.

Návrh

Podobne ako pri lokalizácií môžme skratky definovať v XML súbore. Tým sa zabezpečí perzistovateľnosť a upravovateľnosť skratiek. Je potrebné vytvoriť podpornú triedu, ktora bude tento XML súbor čítať, a zároveň bude overovať stláčanie kláves, ktorými sú tvorené klávesové skratky. Pre účely ďalšieho použitia musí pomocná trieda definovať spôsob, ako pretvoriť klávesovú skratku na textový reťazec. Tým sa zabezpečí zobrazovanie aktuálnych skratiek používateľovi.

Implementácia

Funkovanie skratiek je zabezpečené dvoma súbormi: *Hotkeys.xml* a *HotkeyManager.cs*.

XML súbor

V XML súbore sú definované používané klávesové skratky. Elementy majú názov, ktorý sa používa ako kľúč ku klávesovej skratke. Majú nepovinný atribút *modifier*, ktorý sa používa na definovanie dvojklávesových skratiek. Samotná textová hodnota elementu predstavuje konkrétny kláves.

Hodnoty v XML súbore predstavujú *KeyCode* Enum frameworku Unity. Pre definovanie novej klávesovej skratky je ptorebné ako hodnotu zadať celé číslo, ktoré na základe tohoto Enum-u definuje požadovaný kláves. To platí rovnako pre *modifier* aj pre samotný kláves.

Pomocná trieda

Trieda *HotkeyManager* implementuje návrhový vzor Singleton. Tým je zabezpečené to, že počas chodu programu existuje iba jedna inštancia tejto triedy a tým pádom je zabezpečená konzistencia klávesových skratiek vo všetkých triedach, ktoré klávesové skratky používajú. Pri Inštancovaní triedy sa prečíta XML súbor so skratkami, a tie sa uložia v operačnej pamäti. Na to slúži dátová štruktúra Dictionary, kde kľúčom je textový reťazec - kľúč ku klávesovej skratke, a hodnotou je trieda *Hotkey,* ktorá je vnorená v triede *HotkeyManager*.

Pomocná trieda poskytuje 3 metódy, ktoré na základe kľúča skratky kontrolujú, či je klávesová skratka prvý krát stlačená, dlho stlačená alebo pustená (*CheckHotkeyDown, CheckHotkey, CheckHotkeyUp*)

Tiež poskytuje metódu GetHotkeyLabel, ktorá na základe kľúča skratky vytvorí textový reťazec, ktorý túto skratku reprezentuje. Všeobecný formát tohoto reťazca je *"<modifier> + <key>"*, ak ide o dvojklávesovú skratku a *"<key>"*, ak ide o jednoklávesovú skratku.

Používanie pomocnej treidy

Pomocnú triedu používa každá iná trieda, ktorá potrebuje zisťovať stlačenie klávesovej skratky. Na to najskôr však musí mať definovaný kľúč ku klávesovej skratke. Tento je definovaný ako verejná konštanta triedy, aby kľúč mohol byť dostupný z každého miesta kódu.

3.1.19 Vykreslovanie čiar [Bentley, Cadillac]

Analýza

Elektrotechnické súčiastky sa v reálnom svete spájajú káblami. V EduSim simulácií tieto káble budú reprezentované vykreslovanými čiarami medzi konektormy súčiastok. Unity API poskytuje komponenty, ktoré dokážu medzi dvoma bodmi vykresliť čiaru rôznej farby či hrúbky. Jedným z takýchto komponentov je *LineRenderer*. Podľa unity dokumentácie potrebujeme pridať prázdny *GameObject* a v ňom zahrnúť LineRenderer komponent. Čiara by sa mala začať vykreslovať po kliknutí na niektorý z konektorov a pri držaní stlačeného tlačidla na myši by koncový bod čiary mal

mať súradnice aktuálnej pozície myši. Na detekciu kliknutia preto potrebujeme pridať konektorom *Collider* komponent. Čiara by sa mala vykresliť len v prípade, že sa úspešne spoja dva konektory. Taktiež po spojení dvoch súčiastok sa musí aktualizovať zoznam prepojení, aby sa dala správne implementovať logika elektrického obvodu.

Návrh

Po pridaní Colliderov ku konnektorom a vytvorení objektu zahrňujúceho komponent LineRenderer implementujeme dva scripty. Jeden script s názvom *Line.cs* bude pridaný ku komponentu s LineRenderom a druhý script s názvom *Connectable.cs* ku každému konektoru súčiastky. Connectable script bude obsahovať public atribút, ktorý bude obsahovať list čiarou pripojených konektorov ku konkrétnemu konektoru. Collider konektoru detekuje kliknutie myši na daný konektor. Následne sa vytvorí klon objektu s LineRenderom (každá čiara budem mať vastný LineRenderer). Connectable script zaznamená pozíciu konektora a myši a tieto pozície pošle scriptu Line, ktorý medzi nimi vykreslí čiaru. Po skončení dragovania myšou sa overí, či sa myš nachádza na pozícií nejakého z konektorov. Je potrebné zabrániť spojeniu konektorov tej istej súčiastky alebo konektora samého zo sebou. Vykreslenej čiare je potrebné pridať dynamický Collider, ktorý mení veľkosť a pozíciu pri posúvaní pripojených súčiastok. Tento Collider bude slúžiť na select čiary a následnú možnosť vymazania čiary po stlačení klávesy delete. Implementovaná bude tiež možnosť zmeniť typ selektnutej čiary stlačením klávesy space. Čiara sa bude zalamovať dvomi spôsobmi - pravé zalomenie a ľavé zalomenie.

Implementácia

Ťahanie čiar

Connectable script obsahuje funkcie:

- Start(),
- AddConnected (GameObject connected)

a funkcie rozhraní:

- IBeginDragHandler - OnBeginDrag (PointerEventData eventData) ,
- IDragHandler - OnDrag (PointerEventData eventData),
- IEndDragHandler - OnEndDrag (PointerEventData eventData).

Start()

Funkcia *Start()* inicializuje list objektov, do ktorého sa pridávajú konektory po spojení čiarou a volá sa vždy len pri spustení programu.

AddConnected (GameObject connected)

Funkcia *AddConnected (GameObject connected)* slúži na pridanie druhého konektora do listu (nie toho, ktorého skript sa vykonáva). Využíva unity metódu *SendMessage (message, object)*, ktorá zavolá v *objecte* metódu s názvom *message.*

OnBeginDrag (PointerEventData eventData)

Funkcia zachytáva event začiatku ťahania myšou - stlačenie ľavého klávesu na GameObject, ku ktorému je script priradený. V tejto funkcii sa vytvorí inštancia *Line* objektu, ktorému sa nastaví pozícia konektora ako začiatočný a zároveň aj konečný bod pre vykreslovanie čiary.

OnDrag (PointerEventData eventData)

Táto funkcia je volaná od prvého stlačenia ľavého tlačidla myši až po pustenie tlačidla. V tejto funkcii sa aktualizuje konečná pozícia čiary podľa pozície myši.

OnEndDrag (PointerEventData eventData).

Táto funkcia sa volá v momente pustenia ľavého tlačidla myši. Funkcia najskôr porovná aktuálnu pozíciu myši so všetkými konektormi v scéne. Ak sa pozícia myši s niektorým konektorom zhoduje a sú splnené podmienky, že začiatočný aj konečný bod sa nachádzajú v scéne, spojené konektory nepatria jednej súčiastke a konektor nie je spájaný sám so sebou, aktualizujú sa zoznamy pripojených konektorov v konektoroch a vytvorí sa konečná inštancia čiary medzi konektormi. Ostatné dočasné inštancie (tie, ktoré nemajú koncový bod) sú zmazané.

Line script obsahuje tieto funkcie:

- Update(),
- CheckSelect(),

a funkciu rozhrania:

- IPointerClickHandler - OnPointerClick (PointerEventData eventData),

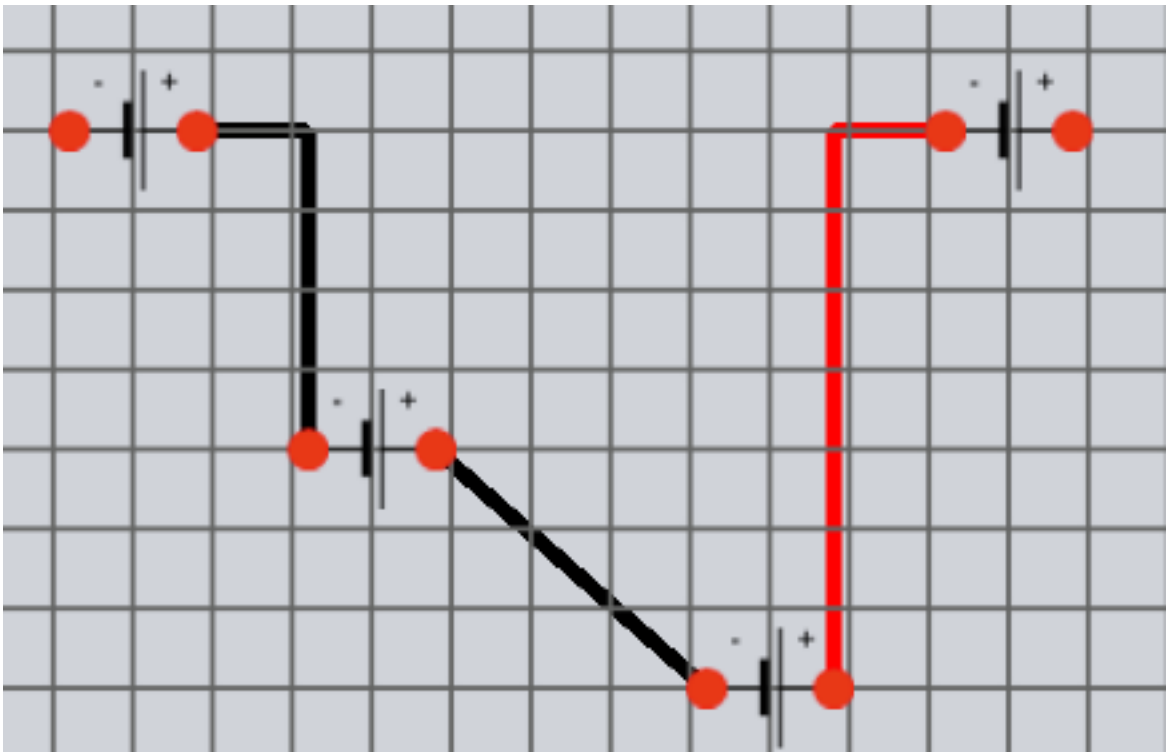
Update()

Táto funkcia sa volá každý frame. Connectable script jej posiela začiatočný a koncový bod - object, medzi ktorými má byť vykreslovaná čiara. Vo funkcii sa z týchto objectov získajú súradnice. Funkcia zavolá komponent *LineRenderer,* do ktorého pridá dané súradnice.

Funkcia overuje, či boli spojené dva konektory. Obsahuje privátnu premennú *TypeOfLine,* ktorá nadobúda hodnoty:

- *NoBreak*, čiže čiara nie je zalomená, a teda ide priamo medzi konektormi.
- *RightBreak,* čiže pravé zalomenie.
- *LeftBreak,* čiže ľavé zalomenie

Pre lepšie predstavenie, čím sa myslí pravé a ľavé zalomenie - ukážka na obrázku nižšie.



V obrázku na ľavej strane je pravé zalomenie, v strede je čiara bez zalomenia a vpravo je zas ľavé zalomenie. Tieto režimy sa prepínajú stláčaním kláves *space*. Obrázok tiež znázorňuje ako vyzerá selektnutá čiara - červená.

Ďalej sa vo funkcii zavolá funkcia AddCollidersToLine(), ktorá čiare pridá komponent *BoxCollider2D*. Funkcia tiež overuje, či sa so súčiastkou nehýbalo alebo či sa nezmenil štýl zalomenia. V tom prípade treba Collider aktualizovať, pretože čiara mení svoju polohu na ploche. Aktualizácia Collidera prebieha tak, že starý Collider sa odstráni a vytvorí nový.

Na konci sa ešte zavolá funkcia *CheckSelect()*, ktorá zisťuje, či bolo na čiaru kliknuté myšou a podľa potreby ju selectne alebo deselectne.

OnPointerClick (PointerEventData eventData)

Táto funkcia detekuje kliknutie na čiaru pomocou eventu *eventData*. Ak sa v momente kliknutia pozícia myši nachádza na *BoxCollider2D*, ktoý patrí danej čiare, čiaru zafarbí na červeno a do globálnej premennej *SelectedLine* uloží objekt selectnutej čiary. Funkcia tiež overuje, či predtým nebola selectnutá iná čiara. Ak áno, predtým selectnutú čiaru najprv vyfarbí naspäť na čierno. Selektnutú čiaru je možné klávesou *delete* vymazať, alebo stláčaním klávesy *space* meniť zalomenie čiary.

AddColliderToLine()

Táto funkcia vytvára BoxCollider2D pre každú čiaru. Collider je pridaný ako child do konkrétnej čiary (do objektu Line). V prípade, že čiara nie je zalomená, zistí sa dĺžka čiary a collider sa umiestni do jej stredu. Funkcia X-ovú dĺžku collidera nastaví na celú dĺžku čiary a y-ovú dĺžku nastaví na veľkosť 0.3, pretože čiara je hrubá len 0.1 a na tak úzku čiaru by sa ťažko klikalo. Nakoniec sa matematicky zistí uhol rotácie collidera, podľa začiatočného a konečného bodu čiary.

V prípade zalomenej čiary sa vytvárajú dva Collidery. Využíva sa pozícia bodu zalomenia. Zistia sa dĺžky čiary od oboch konektorov k bodu zalomenia a funkcia nastavý šírku a dĺžku Collidera podľa toho, či je čiara zvislá alebo vodorovná.

SwitchTypeLine.cs script slúži na spomínané identifikovanie typu zalomenia čiary. Skript v *Update()* funkcii detekuje každách 25 stotín sekundy stlačenie klávesy *space*. Po stlačení klávesy nastavuje premennú *TypeOfLine* v skripte *Line.cs*.

Select a deselect čiar

Select čiar je vykonávaný vďaka BoxColliderú, ktorý je dynamicky pridaný každej čiare. Select je implementovaný vo funkcií *OnPointerClick (PointerEventData eventData)* v scripte *Line.cs,* ktorá je popísana vyššie v dokumentácií. Ak sa myšou klikne na plochu mimo, čiara sa deselectuje - vyfarbí sa na čierno a globálna premenná *SelectedLine* sa nastaví na null. Deselect je implementovaný v scripte *Deselect.cs*. Tento script je pridaný každému objektu *Line*.

Deselect čiar sa vykonáva v scripte *Deselecs.cs*. Tento script je pridaný každému objektu Line. Scrtipt obsahuje už spomínanú funkciu *OnPointerClick*, ktorá sleduje event kliknutia myšou na scénku. Po kliknutí na scénku mimo collidera čiary sa najprv overí, či je nejaká z čiar selectnutá, čiže globálna premenná *SelectedLine* nie je *null*. Ak premenná obsahuje object čiary, danej čiare nastaví farbu na čiernu a premennú *SelectedLine* nastaví na hodnotu *null*.

Mazanie čiar

Mazanie čiar je implementované v scripte *Destroy.cs*. Tento script je pridaný každému objektu *Line*. Ak je nejaká čiara selectnutá (bolo na ňu kliknuté myšou a má červenú farbu), v globálnej premennej *SelectedLine* je uložený objekt tejto čiary. Destroy script overuje, či premenná SelectedLine obsahuje nejakú čiaru. Ak áno, zisťuje, či bola stlačená klávesa *delete*. Po stlačení tejto klávesy sa zo zoznamov pripojených konektorov dané konektory vymažú a nakoniec sa vymaže aj konkrétny objekt čiary.

3.2 *Simulačná logika*

3.2.1 Súčiastky [Audi]

Analýza

Na základe internetového prieskumu boli zistené základné informácie o fyzikálnych veličinách v obvode a súčiastkach v ňom.

Návrh

Na základe prieskumu webu bolo identifikovaných niekoľko elektrických súčiastok a ku každej bolo identifikovaných niekoľko základných vlastností:

- **Rezistor** – odpor, tolerancia odporu(odchýlka k odporu), zaťaženie odporu(dovolený výkon premeny v teplo), dovolené napätie(najväčšie dovolené napätie medzi vývodmi súčiastky), teplotný súčiniteľ(zmena odporu pri zmene teploty o 1 stupeň)
- **Kondenzátor** – kapacita(veľkosť elektrického náboja pri jednotkovom elektrickom napätí), maximálne povolené napätie

- **Cievka** – indukčnosť(množstvo magnetického toku vyvolaného elektrickým tokom), práca(vykonaná cievkou po odpojení prúdu kým dosiahne nulovú hodnotu)
- **Akumulátor(sekundárny článok)**
- **Batéria(primárny článok)**
- **Spínač**
- **Žiarovka**

Takisto boli na základe používateľských prípadov použitia identifikované niektoré generické komponenty, ktoré sa budú nachádzať v každom vzdelávacom module.

- **Stlačitelný bod**
- **Vyskakovacie okno**
- **Merač**
- **Bod merania**
- **Textové pole**

Implementácia

Súčiastky boli vytvorené ako prefabs s tým, že im boli zo začiatku priradené obrázky z internetu a neskôr boli tieto vymenené za obrázky poslané ATOS-om. Každému prefab súčiastky bol priradený skript jej triedy, ktorá bola vytvorená s identifikovanými atribútmi z návrhu. Bola vymyslená schéma pripájania súčiastok a to použitím jednotnej triedy *Connector,* ktorá reprezentuje konektor súčiastky, alebo uzol, ktorý sa pripája k ďalším súčiastkam, alebo uzlom. Každý prefab súčiastky má takisto dva konektory, ktoré majú priradený skript triedy Connector. Prepojenie dvoch konektorov je reprezentácia drátu a je uskutočnené pridaním jedného konektoru do zoznamu pripojených konektorov druhého konektoru, pričom zoznam konektorov je ako atribút v každej triede Connector, a takisto pridaním druhého konektoru do zoznamu pripojených konektorov prvého konektoru. Plusy tohto zapojenia sú absencia drátov a možnosť využitia polymorfizmu nakoľko má všetko, čo je zapojené, nadtriedu Connector.

3.2.2 Back-End simulácie elektrického obvodu [Bentley]

Analýza

Pre potreby výpočtov vlastností elektrického obvodu ktorý si používateľ vytvorí, je potrebné mať k dispozícii nástroj ktorý dokáže simulovať najrôznejšie zapojenia elektrického obvodu.

Nakoľko vytváranie takéhoto simulačného nástroja vyžaduje výbornú znalosť elektrotechniky a bolo by časovo náročné sme sa rozhodli hľadať riešenia ktoré by bolo možné integrovať do nášho projektu.

Integrácia akejkoľvek knižnice alebo programu do nášho projektu EduSim kladie nároky na vzájomnú komunikáciu aplikácií. Navyše Edusim projekt je postavený na Unity 5.4, ktoré zo sebou nesie vlastnú implementáciu C# framework, ktorý navyše implementuje len verziu .net frameworku 3.5.

Jediným riešením napísaným v C# ktoré sa nám podarilo nájsť bol projekt SharpCircuit prístupný na githube - https://github.com/Mervill/SharpCircuit. Licencia tohoto riešenia je MIT/Boost C++.

Pre integráciu v projekte EduSim, je potrebné poskytnúť použiteľnú knižnicu alebo akékoľvek API ktoré by bolo možné integrovať do projektu pre využitie v simulácií.

Návrh

Nakoľko v samotnom repozitári projektu je ukážka použitia projektu na simulácií s dvomi rezistormi a baterkou, je možné odčítať približné použitie projektu SharpCicuit.

Implementácia

Z projektu SharpCircuit sme vytvorili C# class-library – DLL knižnicu, ktorú sme preložili pre Unity 3.5 framework.

Testovanie

Testovanie sme robili v dvoch fázach

Testovanie funkčnosti SharpCircuitProjektu

Pri tomto testovaní sme zisťovali či výpočty ktoré vykonáva projekt, sú zhodné so skutočnými fyzikálnymi vlastnosťami elektrického obvodu.

Testovanie integrácie do Unity projektu

Týmto testovaním sme objavili prvé problémy ktoré nastali ak DLL bola vytvorená .net frameworkom 4.5. Zistili sme že takto preloženú DLL nie je možné použiť pre Unity projekt

.

3.2.3 Grafické prvky súčiastok - napojenie na logiku [Bentley, Cadillac]

Analýza

Projekt EduSim je vytváraný v prostredí Unity. Knižnica ClassLibrarySharpCircuit ponúka objekty ktoré dokážu vytvoriť simuláciu, avšak nie pre grafické prostredie unity, ale pre konzolový výpis. Nakoľko používateľ si potrebuje simuláciu vytvoriť v grafickom prostredí programu EduSim je potrebné nemapovať objekty z DLL knižnice na grafické objekty simulácie.

V zásade sú potrebné dva typy mapovania

- Mapovanie jednotlivých objektov a ich prepojení na reprezentáciu v simulácií z DLL
- Mapovanie spúšťania simulácie z EduSim programu na spúšťanie simulácie v DLL knižnici ktorá vypočíta simuláciu

Nakoľko sa nám nepodarila nájsť podpora elektrotechnických uzlov v DLL knižnici, nie je možné dávať do simulácie pre DLL uzly, ale je nutné uzly odstrániť a nahradiť ich spojeniami s elektrotechnickými súčiastkami.

Pri mapovaní súčiastok sme identifikovali potrebu zapájať súčiastky s rôznym počtom konektorov (napríklad elektrotechnický uzol má jeden konektor, rezistor má dva konektory, tranzistor má tri konektory).

Jednotlivé elektronické súčiastky môžu mať rôzne vlastnosti (akumulátor napätie, rezistor odpor atď.). Tieto atribúty súčiastok je potrebné mať možnosť pred spustením simulácie meniť.

Grafový algoritmus na prechádzanie uzlov

Nakoľko dll knižnica obvodu neobsahuje triedu uzla a umožňuje len priame prepojenie dll-konektorov súčiastok, je potrebné vytvorené uzly v scénke prehľadávať a zistiť tak pre všetky dll-konektory, ku akým ostatným dll-konektorom sú pripojené. Toto bude realizovať algoritmus, ktorého výstup by mal byť tým pádom zoznam dvojíc, z ktorých prvý element bude dll-konektor a druhý element bude list pripojených dll-konektorov ku konektoru v prvom elemente. Na realizáciu programu, ktorý spracuje stav scénky aj s uzlami a vráti požadovaný výstup identifikovaný vyššie, je preto potrebné vytvoriť algoritmus využívajúci grafové prehľadávanie súčiastok a uzlov v scénke za účelom nájdenia prepojení jednotlivých konektorov súčiastok aj za podmienky, že je v scénke v danom prepojení medzi súčiastkami v ceste viacero uzlov.

Návrh

Jednotlivé objekty elektrotechnických súčiastok sú agregované v skript súčiastok zo simulácie. Každá súčiastka bude obsahovať pole konektorov ktoré budú obsahovať odkazy na objekty konektorov príslušnej elektronickej DLL súčiastky. Každá elektronická súčiastka bude dediť od triedy Component, pričom trieda component bude obsahovať konektory zo scény. Konektory zo scény budú pri inicializácií objektu mapované na konektory z DLL knižnice. Takto sa zabezpečí mapovanie DLL konektorov na konektory reprezentované v scéne.

Atribúty súčiastok je potrebné meniť pomocou get/ set metód.

Grafový algoritmus na prechádzanie uzlov

Navrhovaný grafový algoritmus sa podobá deep-first-search. Tento algoritmus je použitý na nájdenie všetkých konektorov súčiastok, ktoré sú priamo a nepriamo(pomocou uzlov) pripojené ku danému konektoru označenej súčiastky. Konektor reprezentuje uzol, alebo konektor akejkoľvek súčiastky a označená súčiastka reprezentuje súčiastku, pre ktorej konektory sa aktuálne zisťujú priamo a nepriamo pripojené konektory. Pred spustením algoritmu musí byť zachovaných niekoľko inicializačných pravidiel:

1. Uchovávanie zoznamu nasledovníkov(ďalších konektorov) u každého konektora(súčiastkového, ale aj uzla)
2. Uchovávanie si zoznamu navštívených a teda už uzavretých konektorov
3. Uchovávanie si zoznamu práve prehľadávaných konektorov
4. Uchovávanie si zoznamu konektorov patriacich súčiastkam
5. Uchovanie si označenia súčiastky

Jednotlivé kroky algoritmu grafového prehľadávania sú nasledovné:

1. Pri navštívení konektora sa tento zaradí do zoznamu prehľadávaných konektorov a zistí sa, či patrí súčiastke. *Ak patrí súčiastke* a táto súčiastka nie je označená súčiastka, tak sa tento konektor pridá do zoznamu konektorov patriacich súčiastkam v prípade, že sa tam už nenachádza. Po jeho pridaní sa odstráni zo zoznamu prehľadávaných konektorov a uloží sa do zoznamu navštívených konektorov. *Ak ale nepatrí súčiastke*, znamená to, že je uzol a prejde sa na krok 2.
2. Navštívia sa všetky konektory daného konektora po jednom za podmienky, že sa nenachádza v zozname navštívených konektorov a ani v zozname práve prehľadávaných konektorov.

Algoritmus na zistenie všetkých pripojených konektorov pre všetky konektory všetkých súčiastok prejde všetky súčiastky a po jednej ich označí za aktuálne prechádzanú. Ďalej pre každú takúto súčiastku prejde všetky jej konektory a na každý zavolá grafový algoritmus.

Implementácia

Pridávanie súčiastok do elektrického simulačného obvodu

V projekte je trieda GUICircuit ktorá agreguje obvod elektronickej simulácie z DLL knižnice. Obvod je reprezentovaný statickou premennou sim. Do tejto simulácie je možné pridávať elektronické objekty kedykoľvek zavolaním GUICircuit.sim.Create<TypeOfElectronicComponent>(). Z pravidla sa to deje pri pridávaní elektronickej súčiastky do scény kedy sa aj mení tag súčiastky z ToolboxItemActive na tag ActiveItem. Tag ToolboxItemActive predstavuje elektronické súčiastky ktoré nie sú v obvode uvažované, ale slúžia len na výber z toolboxu.

Grafový algoritmus na prechádzanie uzlov

Z pohľadu inicializačných pravidiel je zoznam nasledovníkov prítomný v triede *Connector* a je to jej atribút connectedConnectors[]. Trieda Connector obsahuje takisto dll-konektor súčiastky, ktorej patrí, ak patrí nejakej súčiastke. Zoznamy navštívených, prehľadávaných a konektorov patriacich súčiastkam sú implementované pomocou knižnice *System.Collections* a konkrétne jej kolekcii *Stack*.

Celý algoritmus je spustený zavolaním funkcie *untangle*, ktorá berie ako argument zoznam súčiastok scénky a vracia list štruktúr *ConnectionsOfComponent*. Táto štruktúra obsahuje dll-konektor a zoznam dll-konektorov, ktoré grafový algoritmus nájde ako k nemu priamo, alebo nepriamo pripojené. Funkcia *untangle* prejde všetky súčiastky a postupne každú najskôr označí a na každý jej konektor zavolá funkciu *explore* grafového algoritmu. Táto funkcia naplní zoznam konektorov patriacich súčiastkam, ktoré rekurzívne nájde, pričom tieto konektory sú dll-konektory a sú to práve konektory priamo a nepriamo pripojené k danému prechádzanému konektoru označenej súčiastky. Po tejto funkcii sa celý zásobník vyprázdni do štruktúry *ConnectionsOfComponent* a konkrétne do jej druhého atribútu *connectedDllConnectors*. Následne sa všetky tri zásobníky vyprázdnia, aby bol možný aj ďalší beh funkcie *explore*.

Funkcia *explore* je rekurzívna, pričom začína testom konektora s null hodnotou a pokračuje vložením konektora do zoznamu otvorených konektorov. Ďalej testuje, či je to konektor súčiastky a ak hej a ešte nie je v zozname patriacich nejakej súčiastke, tak ho tam pridá. V inom prípade je tento konektor uzol a zavolá sa rekurzívne znova na všetky jeho konektory, pokiaľ nie sú v zozname prehľadávaných, alebo navštívených konektorov. Po týchto krokoch konektor vyhodí zo zoznamu prehľadávaných a vloží ho do zoznamu navštívených.

Elektronické súčiastky

Vždy ako sa pridá nová elektronická súčiastka do scény v ktorej sa vytvára elektrický obvod sa volá funkcia start danej triedy. Nakoľko objektu sa zmení tag z ToolboxItemActive na tag ActiveItem. Takto vieme v metóde Start() identifikovať že pridaná súčiastka bude súčiastka v elektrickom obvode a preto ju pridáme do elektrického obvodu.

Vlastnosti súčiastok je možné meniť pomocou get/ set atribútov tried súčiastok.

Testovanie

Grafový algoritmus na prechádzanie uzlov

Ako testovanie bolo vytvorených pár umelých scénok, súčiastok a ich prepojené konektory spolu s uzlami, ktoré boli predané tomuto algoritmu. Vykonávanie algoritmu bolo sledované pomocou *Debug.Log()* výstupov.

3.2.4 Simulácia el. obvodu v grafickom prostredí [Cadillac][Dodge]

Implementácia

Spustenie simulácie

Každý objekt v scénke je gameobject a k nemu priradené komponenty. Jeden z komponentov je aj jeho skript na ktorého inštanciu sa dá v scénkovom objekte dopytovať funkciou GetComponent<>(). Práve táto funkcia slúži potom na vytiahnutie inštancií komponentov gameobjektov zo scénky, ktoré sa dostanú pomocou funkcie UnityEngine.Object.FindObjectsOfType(typeof(GameObject)), ktorá vráti pole všetkých objektov v scénke. Tieto objekty je však treba ešte odfiltrovať len na súčiastky a preto sa ešte testuje, či majú priradený tag ActiveItem. Všetky takto získané súčiastky sa vložia do zásobník sceneItems.

Po získaní súčiastok zo scénky sa celý zásobník skopíruje do listu súčiastok, ktorý sa predá ako argument funkcii untangle grafového algoritmu, ktorý vráti zoznam štruktúr o veľkosti počtu súčiastok. Tieto štruktúry sa nazývajú spojenia pre komponent a obsahujú pre každú súčiastku zoznam spojení o veľkosti počtu konektorov tejto súčiastky. Zoznam spojení je takisto štruktúra obsahujúca jeden dll-konektor tohto konektoru a zoznam všetkých dll-konektorov k nemu pripojených identifikovaný práve grafovým algoritmom.

Takýto zoznam štruktúr spojení pre komponent sa prechádza vo slučke, pričom sa vo vnorenej slučke prechádza zoznam spojení pre každý konektor tohto komponentu. Nakoniec sa v najvnorenejšej slučke prechádza zoznam dll-konektorov pripojených k danému konektoru a pomocou funkcie sim.Connect sa po jednej položke spojí celý tento zoznam s daným dll-konektorom tohto konektora.

Takto poprepájané súčiastky môžu byť teraz podrobené simulácii pomocou funkcie sim.doTick. Táto funkcia je opätovane spúšťaná vo funkcií update, ktorá sa volá každým snímkom vykresľovania obrazovky.

Simulácia je spustená po stlačení tlačidla "štart", kedy sa vytvorí nový objekt simulácie dll knižnice, načítajú sa všetky objekty zo scénky a tento objekt sa každému z nich predá na inicializáciu ich dll častí. Následovne sa odstránia uzly a príde sa na priame prepojenia súčiastok grafovým algoritmom. Tieto prepojenia sa pospájajú pre daný objekt simulácie. Po stlačení "stop" tlačidla simulácia prestane tick-ať, pričom pre jej spustenie je znova potrebné stlačiť "štart", kedy sa znova vykonajú hore uvedené kroky. Užívateľ teda môže meniť štruktúru obvodu hocikedy, ale len po stlačení "štart" tlačidla sa mu tieto zmeny preukážu aj v simulácii.

3.2.5 Meracie zariadenia [Dodge]

Nakoľko priamo simulačná knižnica nemá podporu žiadnych meracích zariadení, bolo potrebné zariadenia vytvoriť vlastné.

Voltmeter

Zariadenie voltmeter je určené na meranie napätia na súčiastkach obvodu. Mernou jednotkou sú volty [V].

Na implementáciu zariadenia merajúceho napätie sa použil odpor.

Ampérmeter

Zariadenie ampérmeter je určené na meranie prúdu medzi súčiastkami obvodu. Mernou jednotkou sú ampéry [A].

Na implementáciu zariadenia merajúceho prúd sa použil odpor.