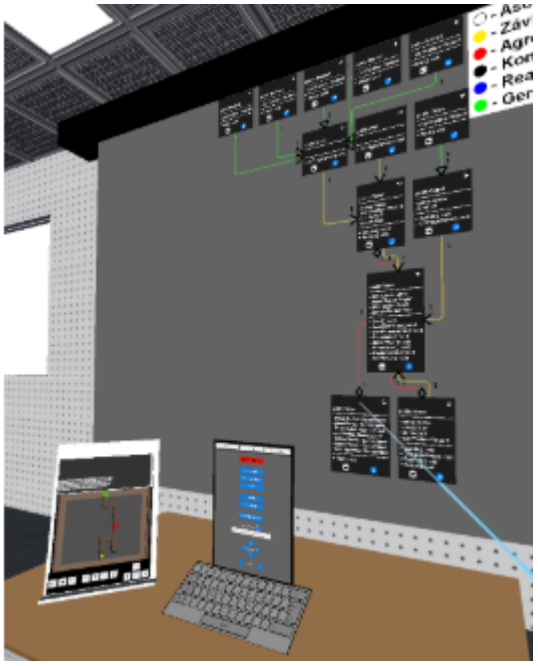
UML-based Live Programming Environment in Virtual Reality

Link

Článok sa snaží priniesť inovaciu do sveta inžinierstva, kde sa moc vecí vo svete klasického programovania neudialo. Jeho prístup je inšpirovaný code bubles. Prichádza s možnosťou úpravy a kompilácie kódu práve pomocou UML diagramov. Jedná sa o class diagramy a sekvenčné ktoré sú navzájom prepojené. Celý program bol spravený v Unity nástroji. Na ovládanie aplikácie bol použitý oculus headset.

Programovacie prostredie pozostáva z 3 častí. Na jednom je plátno na ktoré sa vykresľujú UML diagramy. Diagram je interne reprezentovaný ako graf, kde sú uzly umiestnené podľa layout Sugiyama. Vykreslené boli štandardnou knižnicou Unity MSAGL. Druhej časti sa nachádza menu pre načítanie projektu. Tretej časti človek môže upravovať kód. Pre minimalizovanie ručného písania na klávesnici, boli pridané dva spôsoby a to Cloud speech, ktorý však nebol vhodný pre zadávanie dlhých blokov kódu. Preto bol obmedzený pre zadanie kódu max na jeden riadok. Ďalším spôsobom bolo zadávanie pomocou virtuálnej klávesnice. Pomocou virtuálnych ruk mohol človek zadať kód na virtuálnej klávesnici. Kvôli nekompatibilite medzi použitým kompilátorom Roslyn C# a reprezentáciou VR rúk bol aj tento spôsob zadávania nie úplne dobre fungujúci. Musel byť použitý iný menej presný nástroj na trackovanie rúk.



Najhlavnejšou časťou tohto softvéru je samotná kompilácia kódu z UML na zdrojový a naspäť. Vďaka jeho AST reprezentácii ju vieme pri akejkoľvek zmene upravit a následne zmenu aplikovať vďaka RoslinC# v UML alebo zdrojovom kóde.
Skúmané boli tri otázky:

1) Môže byť vývoj softvéru vo VR (pomocou ovládačov, reči a klávesnice VR) na rovnakej úrovni ako tradičný vývoj softvéru na 2D monitore (pomocou klávesnice a myši?)?

2) Môže vývoj softvéru vo VR zlepšiť používateľskú skúsenosť?

3) Môže vývoj softvéru vo VR skrátiť čas potrebný na kódovanie počítačových programov?

Experimentu sa zúčastnilo 20 ľudí. Každý mal už trochu skúsenosť s UML grafmi. Ich úlohou bolo porovnať tradičné programovanie v Unity s programovaním vo VR. Dostali hru s 3 zadaniami. V každom bolo treba niečo upraviť poprípade dorobiť. Išlo o hru podobnú Super Mariovi. Upravovala sa mu rýchlosť, pridávali prekážky a podobne. Ich užívateľská skúsenosť bola vyhodnotená dotazníkmi SUS pre použiteľnosť, užitočnosť app a UEQ pre užívateľskú skúsenosť osobné dojmy.
VR ide zaznamenalo lepšie výsledky čo sa týka atraktívnosti, prehľadnosti avšak v čase bolo v každom zadaní horšie než štandardný spôsob programovania. Účastníci ocenili prístup písania pomocou virtuálnej klávesnice. Naopak nikto nepoužil zadanie kódu pomocou reči. Ako návrh na zlepšenie bola možnosť zvýraznenia vybranej triedy class diagramu pri jej editácii, takisto možnosť úpravy kódu pomocou gest.

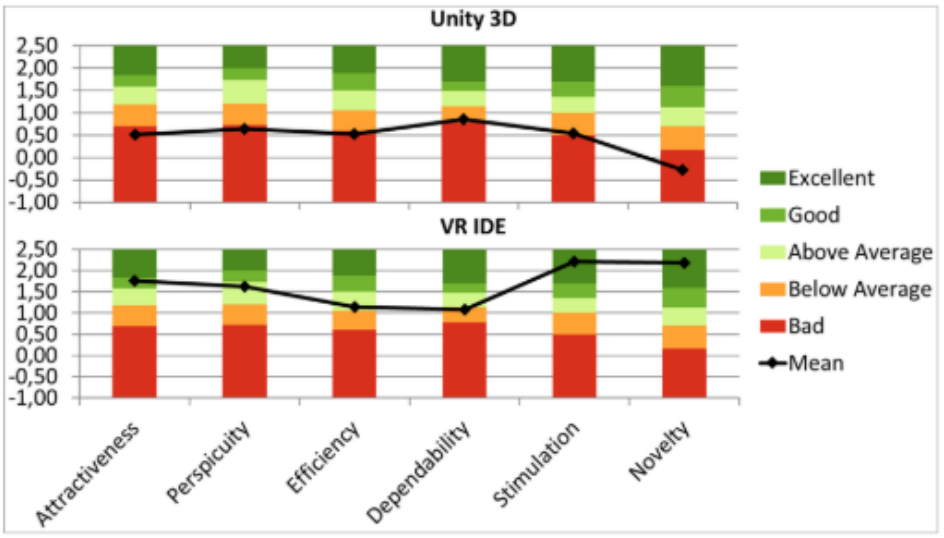


Fig. 4. UEQ VR IDE versus Unity 3D.

Toward a VR-Native Live Programming Environment

Link

Stretávajú sa so strašne veľa prípadmi pri programovaní vo VR kedy musia v istej fáze vývoja aplikácie zložiť z hlavy headset a manuálne nejaké veci urobiť. Preto vyvinuli softvér ktorý umožňuje vo VR programovať skompilovať aplikaciu bez potreby dať si dole okuliare. Pre ovládanie aplikácie bol použitý Headset a hand trackery. Pri vývoji softvéru bola snaha o minimalizovanie písania pomocou mechanickej klávesnice. Boli tu rozobraté rôzne prístupy programovania v related work. Napríklad premietnutie kódu vďaka kamerám headsetu z obrazovky PC do VR na plátno, následne ich editácia pomocou myše a klávesnice. Ich softvér fungoval na systéme Smalltalk ktorý slúžil pre live programovanie. Zobrazenie Smalltalk kódu do 3D priestoru sa realizovalo pomocou syntaktického stromu, ktorý nám vygeneroval z kódu analyzátor. Na interakciu s kódom nám miesto VR ruk poslúžil prútik, ktorý slúžil ako ukazovatko. Syntaktický strom sa zobrazí vo forme blokov (kvádrov) . Časti stromu vieme prútikom chytiť a odložiť si ich poprípade ich premiestniť (drag and drop). Pri nadídení prútikom nad block sa zobrazil jeho krátky popis. Celkovo bolo možné pomocou upravovať štítky labelu ak chce človek vytvoriť nový block začal priamo písať do bloku text a vytvoril si presne typ ktorý potreboval. Nebola zadaná žiadna dopredu ponuka hotových tipov blokov. Písalo sa pomocou virtuálnej klávesnice, alebo sa kreslil text ktorý potom softvér automaticky rozoznáva také ako na google prekladači.

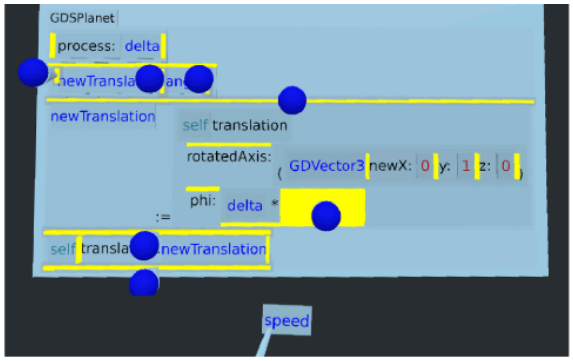


Figure 3. At the top, a collection of blocks forming a method and a single block containing the identifier "speed" are depicted. After picking up the identifier block (below), blue spheres mark possible insert positions in the method.

Po kliknutí na daný objekt triedy sa zobrazí daná metóda samostatne. Človek môže rýchlo efektívne drag and drop metódy presúvať. Celkovo je to live coding čiže akékoľvek zmeny je v metódach hneď vidieť. Ak je v metóde chyba tak si človek vidí chybu, čiže má aj error detection. Keďže fungovalo live programming človek mohol okamžite vidieť zmeny v kóde a to aj pomocou frontendovej aplikacie, ktorá bola vyrobená v Godot engine.
Výskumu sa zúčastnili ľudia ich úlohou bolo bez pomoci návodu dorobiť fungovanie slnečnej sústavy. Nechali ich doprogramovať rotovanie okolo slnka alebo setnut rýchlosť objektov. Po skončení celého experimentu poskytli svoje pripomienky. Počas experimentu boli prítomní aj inštruktori ktorí si všetko merali. Pomohli keď sa účastník aspoň minútu nepohol so zadaním. Prvá časť zadania bola bez vysvetlenia ako sa program používa a to kvôli vyskúšaniu jeho intuitivnosti. Druhom zadaní im už bol poskytnutý krátky návod. Účastníkom chýbalo prezeranie definície tried, metód, takisto sa cítili byť menej produktívni ako pri normálnom programovaní. Celkovo však účastníci ocenili možnosť živého kódovania, a automatickej kompilácie.

Generating Java Code from UML Class and Sequence Diagrams

Link

Výroba softvéru je čoraz časovo zložitejšia kvôli logike softvéru. Preto sa snažia ľudia výrobu softvéru urýchliť a to napríklad MDE. MDE modely sa postupne transformujú dovtedy až kým z nich nie je možné získať implementáciu softvéru. Na tvorbu týchto modelov nám môžu priamo slúžiť aj UML diagramy, ktoré priamo reprezentujú implementáciu softvéru. Bude použitý jeden class diagram a zopár sekvenčných. Získaný kód bude v jazyku Java. Generovanie bude prebiehať nasledovne. Najskôr sa vygeneruje štrukturálny kód z class diagramu (1 class = 1 súbor) a následne sa zo sekvenčných diagramov vytiahnu všetky veci až po úroveň volania metód. Primitívne operácie ako sú priradenia alebo aritmetické operácie nie je možné zachytiť v sekvenčných diagramoch. For loops a switch a ifs nie sú problém. Ako ukážkový príklad bol použitý program s 1 class diagramom a 8 sekvenčnými.

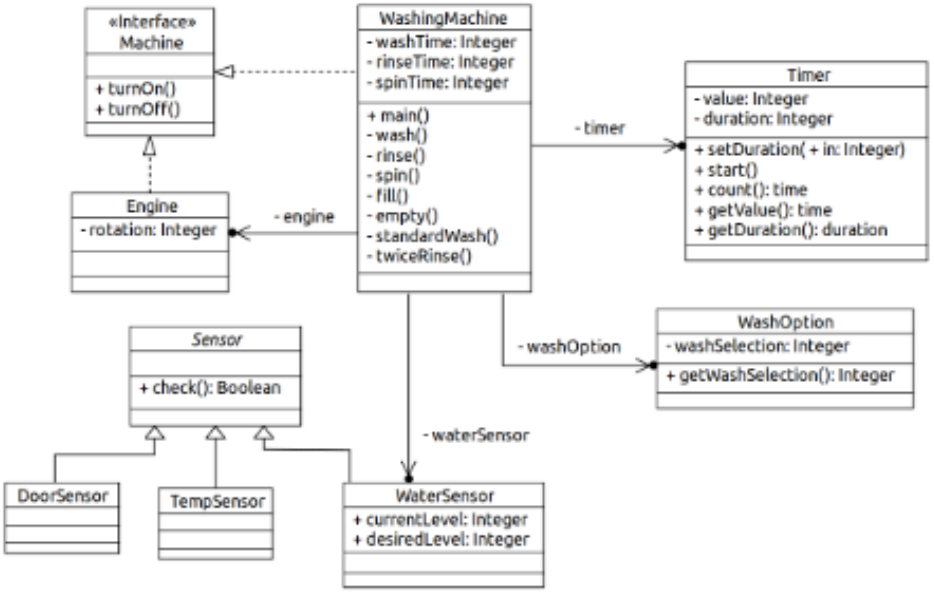


Figure 1. Class diagram - Structural view of Washing Machine.

Príklad zahrňuje aj abstraktnú triedu. Kód sa modeluje od main metody kde to celé začína tak ako v štandardnom kóde. Dole na obrázku je jej sekvenčný diagram.

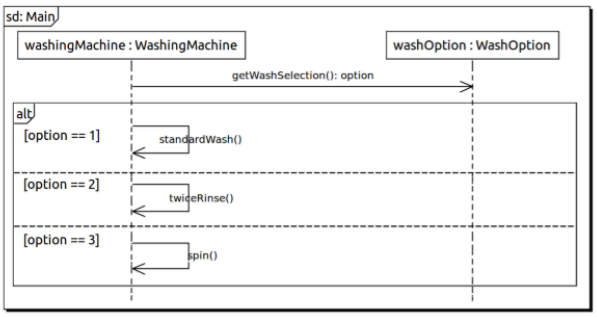


Figure 2. Sequence diagram for the Method Main

Všetky metódy nimi posielané premenné sú zachytené a pridané do kódu.

UML to code, and code to UML

Link

UML je populárne pre vizuálnu reprezentáciu softvéru. Cieľom článku je analyzovať rôzne implementácie na generovanie kódu UML a porovnať ich na základe rôznych parametrov špecifikovanej implementácie, ako je čas kompilácie, cyklomatická zložitosť a iné. Historicky už boli pokusy o generovanie kódu z UML: Visual Paradigm, Software Ideas, StarUML generovanie Java kódu. Výhodou takéhoto spôsobu vývoja softvéru je značne zníženie investicií pri vývoji projektu čo sa času týka. Takisto predídenie návrhovej chybe v architektúre. Úlohou bolo poskytnúť kompletný balík konvertovania z UML do kódu aj naspäť. Parametre, ktoré sú najvhodnejšie na krátku analýzu nástroja UML na kódovanie, sú: konzistencia a správnosť, výkon (CPU aj pamäť) a uspokojiteľnosť. Ďalšou vecou je umelá inteligencia ktorá sa vďaka hlbkovému učeniu zlepšila a dnes je uz schopná lepšie analyzovať zdrojové kódy. Jednym z tychto analyzátorov je DL. Pomocou DL môžeme analyzovať rôzne zdrojové kódy, ktore sa vieme spojiť s modelom a neskôr vytvoriť celú infraštruktúru založený na algoritme učenia.

[2].

Table I. Efficiency for different currently available tools. (adapted from [10])

Tool Name	Classes/scope	No of constraints	Efficiency
UMLtoCSP	Small (2-50)	Small (5-10)	In few minutes
	Large(100-above)	Small (5-10)	Timeout
UML2Alloy	Small (1-7)	Small (5-10)	In few minutes
	Large (100-1000)	Small (5-10)	Timeout
USOT	Large (above 100)	Small (5-10)	Less time required as compared to UMLtoCSP and UML2Alloy
	Large (above 100)	Large (100-above)	Timeout

The parameters that are best suited for a brief analyze of the UML to code tool are: consistency and well-formedness, the performance meaning the computation allocated resource need which include both the CPU and the memory, and the last parameter that is important is the satisfiability which is represented by class liveliness and the constraint of redundancy. As described in [10], in model implementations

Aplikácia má tri hlavné komponenty, používateľské rozhranie, prevodník a komponent sledovateľnosti. Komponenty sú nezávislé a môžeme vďaka tomu meniť ich implementáciu. Už len z toho dôvodu že keď chceme vygenerovať UML z alebo do iného kódovacieho jazyka. Človek je schopný pekne vygenerovať si UML zo zdrojového kódu keď potrebuje a zároveň všeobecne zmena v kóde sa bez problémov prejaví v UML. UML je prezentované XML mapou.

Generovanie kódu z UML prebieha nasledovne. Najskôr sa UML prevedie pomocou XML convertera do XML mapy. Tu converter odchytáva kľúčové slová ID, VAL and IDEND. Potom sa preberajú tieto informácie a vytvoria sa objekty oblasti vlákien s pracovnými vláknami, ktoré prevezmú skonvertované informácie a vytvoria object Type pomocou reflexie. Objekty sa uložia vo Folder Assembleri a sú použité neskoršie knižnicami na generovanie kódu v špecifickom jazyku. Assembler je zodpovedný za písanie kódu. Generátor na generuje z objektov kód. Generátor je pre každý druh kódu iný. Refactoring engine je zodpovedný za kod, ktorý sa vygeneruje z XML convertera.

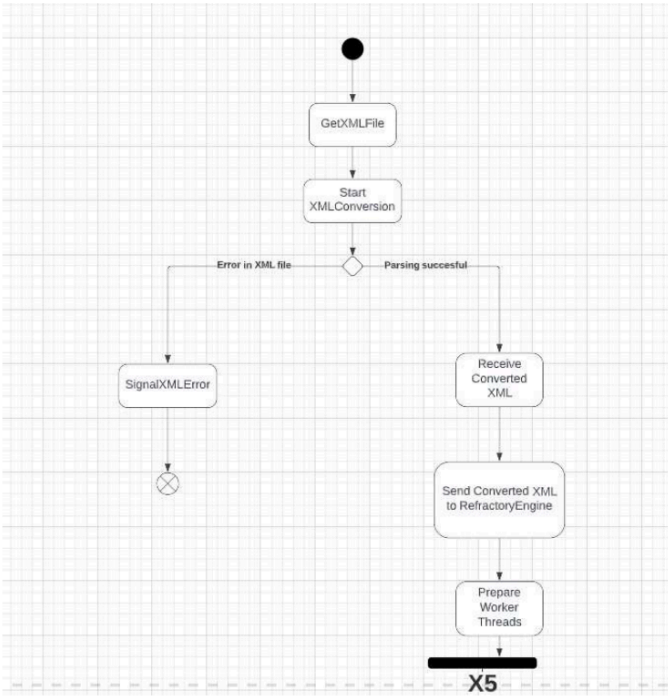


Fig. 4. Activity diagram first page

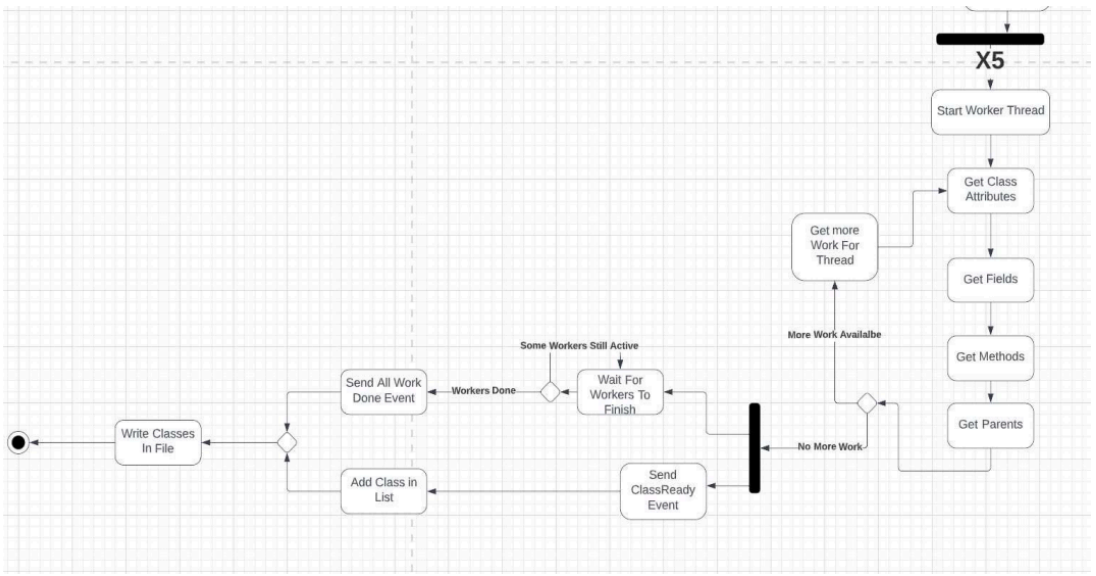


Fig. 5. Activity diagram second page

Ak dôjde nejakej zmene v kóde file listener túto zmenu zachytí a pošle do Code tracera, ktorý rozhodne, či zmena ovplyvní diagram UML. Testovanie súvisiace s aplikáciou bolo robené počtom rôznych tried a porovnávané na rôznych procesoroch. Táto aplikácia zatiaľ umožnila generovanie len Java kódu.

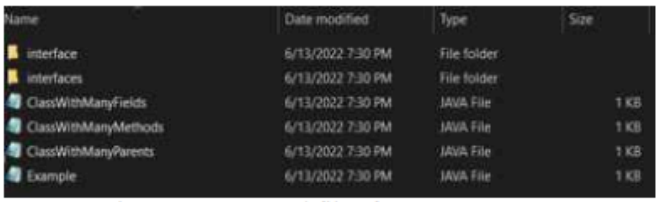


Fig. 8. Generated files from XML

Table II. Experimental results using the resource monitor.

CPU type	Number of classes	CPU consumption average based on time	Time (MS)
I7 11800H	10	0.03	0.028
	100	0.64	0.45
i5 8350U	10	0.64	0.05
	100	2.56	0.75
I7 8550U	10	0.48	0.036
	100	0.7	0.56
AMD Ryzen 7 3700 U	10	0.5	0.03
	100	0.75	0.7

Spoznámkované ostatné vedecké články v odrážkach

1) David Moreno-Lumbreras; Jesus M. Gonzalez-Barahona; Andrea Villaverde - BabiaXR: Virtual Reality software data visualizations for the Web : link
2) Matej Ferenc; Ivan Polasek; Juraj Vincur - Collaborative Modeling and Visualization of Software Systems Using Multidimensional UML : link
3) Enes Yigitbas; Simon Gorissen; Nils Weidmann; Gregor Engels - Collaborative Software Modeling in Virtual Reality: link
4) Juraj Vincur; Pavol Navrat; Ivan Polasek - VR City - Software Analysis in Virtual Reality Environment: link
5) Anthony Elliott; Brian Peiris; Chris Parnin - Virtual Reality in Software Engineering: Affordances, Applications, and Challenges: link
6) Dussan Freire-Pozo; Kevin Cespedes-Arancibia; Leonel Merino; Alison Fernandez-Blanco - DGT-AR: Visualizing Code Dependencies in AR : link
7) Peter Kapec, Juraj Vincur, Miroslav Kozma - CollaVRation: An Immersive Virtual Environment for Collaborative Software Development : link
8) James Dominic; Brock Tubre; Jada Houser; Charles Ritter; Deborah Kunkel; - Program Comprehension in Virtual Reality : link
9) P. Young; M. Munro - Visualising software in virtual reality : link
10) Lucas Kreber; Stephan Diehl; Patrick Weil - IDEvelopAR: A Programming Interface to enhance Code Understanding in Augmented Reality : link
11) Rohit Mehra; Vibhu Saujanya Sharma; Vikrant Kaulgud; Sanjay Podder - XRaSE: Towards Virtually Tangible Software using Augmented Reality : link
12) Enes Yigitbas, Ivan Jovanovikj, Gregor Engels - Simplifying Robot Programming using Augmented Reality and End-User Development : link
13) Leonel Merino; Alexandre Bergel; Oscar Nierstrasz - Overcoming Issues of 3D Software Visualization through Immersive Augmented Reality : link
14) Roy Oberhauser - VR-UML: The Unified Modeling Language in Virtual Reality – An Immersive Modeling Experience : link

15) Steven Reiss, Jared N. Bott, Joseph J. LaViola Jr - Code Bubbles: A practical working-set programming environment : link
16) Rodi Jolak; Khan-Duy Le; Kaan Burak Sener; Michel R.V. Chaudron - OctoBubbles: A Multi-view interactive environment for concurrent visualization and synchronization of UML models and code : link
17) Sara Gotti; Samir Mbarki - UML executable: A comparative study of UML compilers and interpreters : link
18) Victor Stefano Segura Castillo, Leonel Merino, Geoffrey Hecht, Alexandre Bergel - VR-Based User Interactions to Exploit Infinite Space in Programming Activities : link

Spoznámkované články z internetu

1) Ruah Tech - Unreal vs Unity vs Godot : link
2) XR Interaction Toolkit vs Microsoft Mixed Reality Toolkit link1, link2