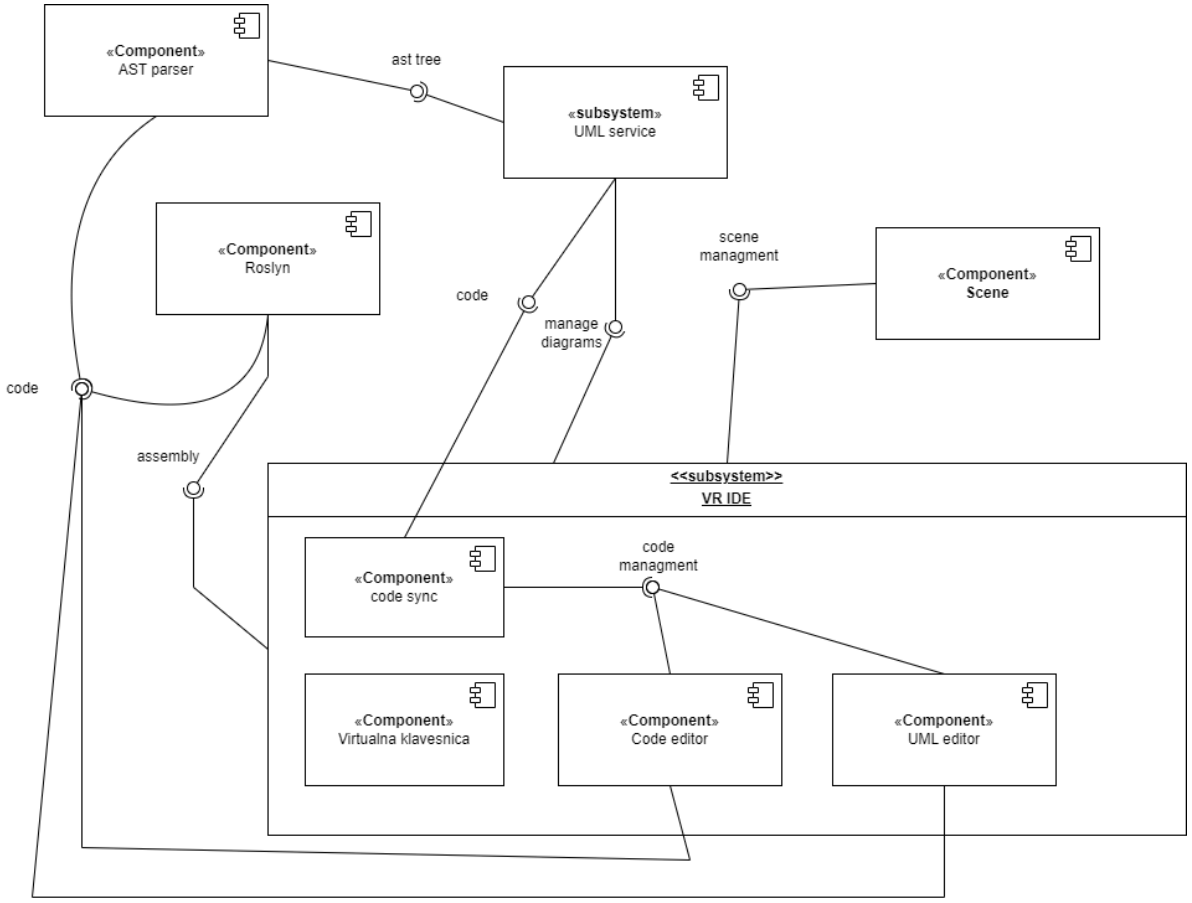
Návrh

component diagram:



funkcionálne požiadavky:

- zobrazenie zakladnej kostry kódu pomocou class diagramu (jeho triedy, prepojenia medzi nimi, metódy, premenné)
- možnosť zobrazenia activity diagramu pre jednotlivé metódy activity.
- možnosť editácie UML pomocou interaktívneho menu
- dodatocna editácia UML pomocou virtualnej klavesnice
- kompilovanie kodu (možnosť vyskúšať si fungovanie aplikácie)
- detekcia chyb (zobrazenie konzoly s chybami)
- generovanie kódu z UML

nefunkcionálne požiadavky:

- čas kompilácie z UML na kód by mal byť pri veľkosti 3 tried 10 metód 2 sekundy
- minimalizácia písania na virtuálnej klávesnici (o 50 percent)
- rozmerovo obmedzený priestor na pohyb vo VR (rozmery)

návrh evaluácie:

Predpoklad:

1) ľudia budú mať zo začiatku štandardné problémy s používaním VR headsetu
2) po pár hodinách by sa programovanie softvéru pomocou VR mohlo zlepšiť (časovo)

Priebeh

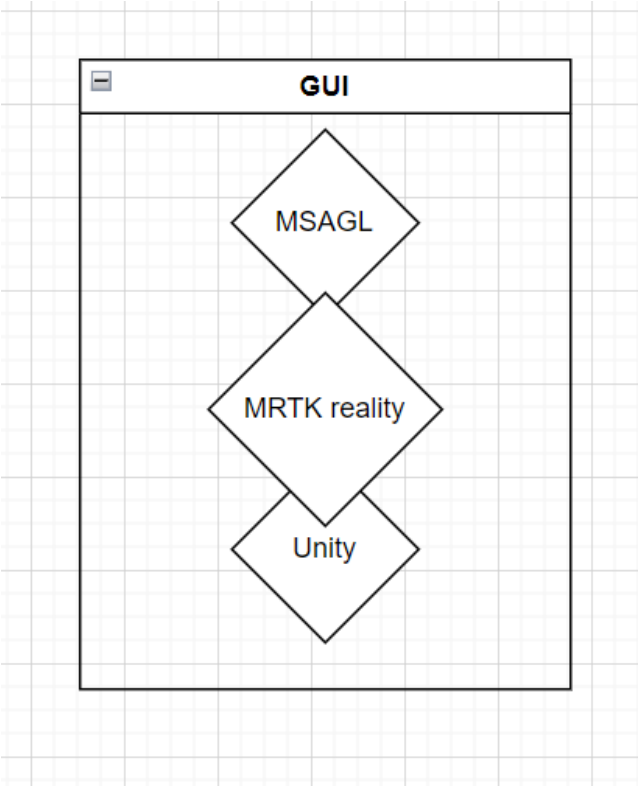
Na začiatku budú účastníci oboznámení s programom jeho fungovanie. Budú si môcť vyskúšať interakciu so scénou vo VR.
Následne ucastnici dostanú 3 programy v UML rôznej obtiažnosti.
V prvom bude musiet človek dorobiť funkcionalitu.
Na druhý program bude človek odpovedať na vopred položené otázky týkajúce jeho fungovania. Preto bude musieť porozumieť ako sa program správa a to na základe UML diagramov (class aj activity).
V treťom programe budú mať ľudia za úlohu nájsť chyby v UML a opraviť ich.
Tieto zadania sa budú robiť s VR headsetom a bez neho len na emulátore
Prieskum bude meraný na cas. Po skončení programovania vyplnia účastnici UEQ dotaznik.
Výskumné otázky:

1) Dokážeme nájsť vhodné specifikovane UML diagramy pre úplné vygenerovanie kódu?
2) Či sa naše riešenie práce vyrovná tradičnému vývoju softvéru?

Ciele:

Cieľom tejto práce je nájsť vhodný spôsob ako z UML class diagramu a activity generovať čo najviac úplný kód. Zároveň je snaha pri vývoji softvéru minimalizovať písanie na klávesnici.

tech stack:



Ako softvér na vývoj aplikacie bude použitý Unity. Nástroj pre vývoj prostredia vo VR bude použité MRTK. Pre vykresľovanie grafov sa použije MSAGL knižnica.

Špecifikácia UML vs code reprezentacia

class diagram:

vzťahy medzi entitami:

generalizácia bude ak nájdeme za názvom Class1 Class2

realizácia bude ak nájdeme za názvom Class1 iný názov ako class povedzme interface a pod

asociácia je ak nájdeme závislosť class1 v class2 ale nevieme ju špecifikovať ako kompozíciu, agregáciu alebo závislosť

závislosť definuje používanie Class1 v nejakej z metód Class2. Neni však definovaná v jej premenných. Príklady použitia Class1 c = new Class1()

agregácia bude ak ju nájdeme Class1 pri inicializácii premennách v class2 a zároveň jej inicializáciu v niektorej z Class2 metód

kompozícia nastane ak bude nájdená Class1 v niektorej z Class2 premenných a zároveň bude v niektorej z metód definovaná s new

objekt:

v názve bude napisaný názov plus typ či ide o class, abstract class, namespace, interfejs
vnútri objektu budú všetky definované metódy, premenné a ich typy

activity diagram

decision node je jeden node obsahujúci podmienku z neho ide šipka k <> tvaru a stadiaľ idú následne action šipky ďalej if condition -<>->
while, for cycle: je definovaný na začiatku decision node z ktorého idú viaceré šípky. A takisto do decision nodu ide aspoň jedna šípka z nejakého nodu teda príkazu. Vtedy vieme že ide o loop.
v praxi keď budem generovať bude decision node overím či má loop back a na základe toho budem indikovať že je to for/while cycle. Do kódu budem premietať ako while cyklus.
Čiže decision node vetva **no** ide z cyklu von buď koniec metody alebo dalsia akcia. Tu si poznacim. Vetva **yes** pisem do vnutra while(condition){dalsie akcie}
netreba zabudnut pri loope na prípadný increment
if, else,case: decision node mam podmienku a na zaklade odpovede nasledujúcu akciu
No vetva idem z podmienky von na ďalšiu akciu
Yes vetva vnorim sa do podmienky a robim dalsie akcie,
podmienka je jedna a tá istá keď je zložená konjunkcia alebo disjunkcia

prepojenia sú teda dve medzi akciami (príkazmi)
activity edge prepája klasicky dva nody medzi sebou, vtedy to znamená že príkazy nasledujú po sebe
decision node spája podmienkovy node s action nodmi ďalšími poprípade endom či start nodom