УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

ДИЗАЈН И ИМПЛЕМЕНТАЦИЈА СОФТВЕРА „ЛЕЊИР И ШЕСТАР“ И ЊЕГОВО КОРИШЋЕЊЕ У ЦИЉУ УНАПРЕЂЕЊА НАСТАВЕ МАТЕМАТИКЕ

Дипломски мастер рад

Ментор:
Др Владимир Филиповић

Студент:
Дарко Максимовић

Београд, 2012.

Садржај

Увод

У овом раду биће описан развој софтверског производа који може представљати додатак школској настави планиметрије која се изучава у узрасту од петог до осмог разреда основне школе. Пројекат се ослања на претпоставку да је добро да се прошири самостални ваннаставни рад ученика код куће и убјеђење да рачунарска наука може помоћи наставницима да лакше подстакну ученике на то. Захтјеви који ће бити постављени пред софтвер процијењени су након пажљивог проучавања службеног плана и програма наставе математике за поменути узраст, као и основног упознавања са тренутним стањем система образовања у Републици Србији по питању могућности дигитализације наставних средстава и подршке која је тренутно пружена самосталном ваннаставном раду.

Софтверски производ о коме је ријеч добио је назив „Лењир и шестар“, јер су то алати који се највише користе у области планиметрије на овом узрасту, а истовремено кориснички интерфејс је конципиран на такав начин да подсјећа на коришћење ових алата. Ово посљедње је тако осмишљено намјерно, јер педагошка вриједност овог софтвера треба да лежи управо у допуни класичне наставе. „Лењир и шестар“ је сачињен од двију компонената: дизајнера и прегледача. Дизајнер је намијењен наставнику, који у њему може описати одређени геометријски поступак, снимити на преносиви медијум или дијељено мјесто на интернету, те понудити својим ученицима. Ученицима су на располагању и прегледач и дизајнер; помоћу прегледача они могу учитати поступак на свом рачунару код куће, прегледати га и преслушати (аудио компонента је обухваћена програмом да би наставник могао снимити гласовни коментар), а потом потенцијално и учитати у дизајнер, модификовати у циљу увиђања међусобних односа објеката или покушати урадити цијели процес испочетка због вежбе.

Дјелокруг програма је свјесно ограничен уско, и то на неколико начина:

- Обухваћено је наставно градиво само за област планиметрије
- Обухваћен је само узраст од петог до осмог разреда ОШ
- Сам кориснички интерфејс има ограничен скуп алата који би се могао проширити

Разлог за овако узак дјелокруг лежи у томе што цио пројекат, а нарочито овај документ као један његов дио, треба да послужи првенствено као смјерница и илустрација како може да тече свеукупни развој – како проучити потребе, начине њихових задовољења, затим одабрати методологију рада и поставити темељ производа. У овом раду имплементација је потом описана до краја, уз изводе из изворног кода. У прилогу је дат сам софтвер (изворни код и извршна верзија за оперативни систем Виндоуз), као и неколицина датотека које су резултат самог програма и спремне су за учитавање и демонстрацију дотичног геометријског поступка.

Анализа текућег стања и захтјеви који се постављају пред софтвер

Стање система образовања у Србији као платформа за даљи развој

У извјештају о тренутном стању кључних обиљежја система образовања у Србији, у оквиру документа који је израдио Савјет пројекта за израду Стратегије развоја система образовања у Србији до 2020. године (у даљем тексту, СРОС; 2011), као један од главних изазова ове стратегије наводи се да постоји „изузетно мала заступљеност модерних облика рада у школи; наставници нису обучени за примјену модерних концепата учења/наставе и нове улоге која слиједи из њих“. Иако с правом можемо претпоставити да се у овом контексту под *модерним* концептима учења/наставе на првом мјесту подразумијевају повезивање теорије и праксе, експеримент, уопште активно учење и друга достигнућа модерне педагогије, данас се у ову групу незаобилазно сврстава и примјена рачунара и рачунарских технологија у процесу унапређења школске наставе. У Србији овај вид наставног средства још није узео пуног маха највише из материјалних разлога, јер многе школе, посебно у неразвијеним крајевима, још увијек немају ни рачунаре, ни интернет конекцију, а немају ни ученици код куће (СРОС, 2011). Ипак, посматрајући извјештаје Републичког завода за статистику за 2010. годину, долазимо до охрабрујућег податка да је тренд опскрбљивања школа и домаћинстава рачунарским средствима у Србији узлазан, па је тако 2010. године 50,4% домаћинстава посједовало рачунар, што је за 3,6% више у односу на 2009. годину. Такође, у истом извјештају сазнајемо да је 39% домаћинстава посједовало интернет прикључак, што је за 2,3% више у односу на 2009. годину. Такође, владине и не-владине организације повремено позитивитно утичу на овај тренд, као што је у току школске 2008/2009. године Министарство за телекомуникације и информатичко друштво организовало акцију поклањања рачунара школама. Ови подаци наводе нас на закључак да ће у будућности број домаћинстава и школа у Србији које посједују рачунарска средства бити све већи, те да је држава постала довољно зрела да отпочне озбиљније напоре у унапређењу школства користећи се овим средствима, тј. да развије или подржи развој неопходних софтверских пројеката који би то омогућили.

У вези са наставницима и другим наставним особљем, статистички подаци показују да ту већ постоји довољна подршка. Наиме, према извјештајима везаним за стратегију менаџмента људским ресурсима Европе и Америке, велика је заинтересованост за образовање и обуку подржане технологијама е-учења. Постоји више разлога за то: од подизања нивоа ефикасности и квалитета обуке, до превазилажења двају ограничавајућих чинилаца, а то су вријеме и простор.

У оквиру наставе, потенцијалне примјене рачунара су многобројне: од простих презентација на рачунару које се могу припремити само једном и потом представљати ученицима на часу путем великог екрана или пројектора, до интерактивних апликација које могу помоћи ученицима да кроз сопствени рад у апликацији стекну ново знање или обнове већ стечено.

Могућности развоја наставе математике уз помоћ рачунара

Настава математике је нарочито повољна да буде подржана рачунарским средствима. За ово постоји више разлога:

1. У поступцима који се усвајају битно је схватити и запамтити редослијед корака, а не само статичку слику, што може бити потпомогнуто динамиком коју нам пружа рачунарски софтвер;
2. Настава математике обилује графичким приказима и симболима, које различити професори различито приказују цртајући на табли, док се исти униформно приказују и лакше пишу и виде на рачунару;
3. Постоји велика популарност рачунара код дјеце због различитих компјутерских игрица и разноврсног мултимедијалног садржаја које пружа интернет, те се коришћењем рачунара може превазићи отпор који одређени број дјеце гаји према математици, а који између осталих фактора узрокује недовољну математичку и научну писменост код дјеце. (CPOC, 2011)

По питању могућности и стручности за унапређење наставе кроз употребу рачунарских средстава, наставно особље потекло с неког од математичких факултета у Србији налази се у значајној предности у односу на колеге с већине других факултета у Србији, јер се по самој природи ових факултета студенти у току студија упознају са напредним концептима информатике, од коришћења до програмирања истих. Додатно, велики број њих посједује рачунар, јер им је био потребан да би савладали то градиво. Ово пружа увјерење да би професори математике с лакоћом усвојили коришћење нових производа који би им се понудили да би боље и лакше обавили свој посао, уз истовремено побољшање резултата ученика.

У овом раду понудићемо рјешење за помоћ наставницима математике и њиховим ученицима при савлађивању планиметрије – области математике која заузима значајно мјесто у основно-школској настави, те се интензивно проучава цијеле три школске године, од петог до седмог разреда. Биће предочено како се може конципирати, организовати, остварити и спровести један такав пројекат, који ће омогућити наставницима да уз минималан сопствени напор омогуће додатну и допунску наставу за све оне којима је она потребна.

Постоји више циљних група ученика којима је потребна допунска или додатна настава. Набројаћемо само неке од њих:

1. Дјеца с поремећајем пажње потребују допунску наставу (поремећај пажње често је удружен~~а~~ са хиперактивношћу или је узрокован~~а~~ њоме). Статистике показују да се број дјеце са поремећајем пажње удвостручује на сваких 10 година, те да је данас између 7 и 10 процената деце, узраста од 5 до 18 година, са овом дијагнозом (Јовановић, 2005).
2. Дјеца са физичким потешкоћама за редовно праћење наставе
3. Умно надарена дјеца заинтересована за додатну наставу, у циљу стицања знања и вјештина које не улазе у редовни програм школске наставе
4. Дјеца која из различитих разлога потребују индивидуализирану наставу

Циљеви и задаци наставе планиметрије за основну школу

Да бисмо успјешно спровели пројекат подршке какав смо најавили у претходном

поглављу, неопходно је детаљно се упознати са планом и програмом наставе за разреде у току којих се изучава планиметрија. Садржај овог поглавља представља извод из садржаја званичног Плана и програма математике за основну школу Министарства образовања Републике Србије (у даљем тексту, План и програм) и биће ограничен на област планиметрије.

Задаци математике у петом разреду

План и програм предвиђа да ће ученици петог разреда основне школе стећи знања неопходна за разумијевање квантитативних и просторних односа и законитости у разним појавама у природи, друштву и свакодневном животу. Задатак наставе математике у овом разреду, по питању планиметрије и геометрије уопште, јесте да се ученици упознају са најважнијим равним геометријским фигурама и њиховим узајамним односима. Напослијетку, ученици овог разреда треба да буду оспособљени за прецизност у мјерењу, цртање и геометријске конструкције.

Конкретни оперативни задаци планиметрије помоћу којих је у Плану и програму осмишљено стицање поменутих знања и вјештина, јесу сљедећи:

- Геометријске фигуре (права, дуж, полуправа, раван, кружница, круг, угао идр.) уз њихово адекватно описивање
- Углови уз трансверзалу паралелних правих
- Углови са паралелним крацима и њихова својства
- Конструкција праве паралелне задатој правој
- Осна симетрија и њена својства
- Конструкција симетрале дужи, угла и нормале на дату праву кроз дату тачку

Задаци математике у шестом разреду

У оквиру скупа оперативних задатака наставе математике у шестом разреду основне школе, нагласак је стављен на усвајање скупова цијелих и рационалних бројева, као и на стицање потребних знања и вјештина потребних за рјешавање једначина и неједначина. Међутим, и у овом разреду се изучавају основе геометрије равни, па тако у Плану и програму за овај разред срећемо сљедеће:

- Класификација троуглова и четвороуглова и њихова основна својства
- Релација подударности и њена основна својства
- Једнакост површи геометријских фигура и правила о израчунавању површина троуглова, паралелограма и других четвороуглова

Задаци математике у седмом разреду

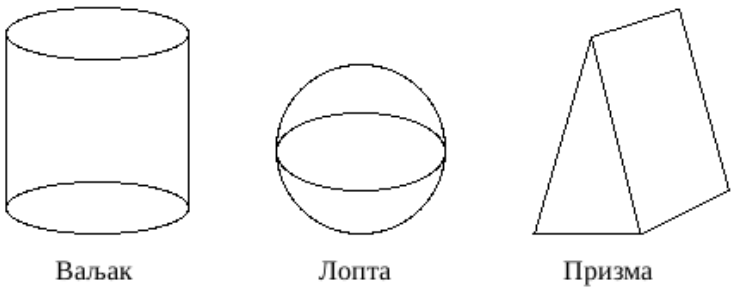
У седмом разреду ученици се упознају са концептом степена броја, те основама полиномијалног рачуна, нешто што ће интензивно упознавати у средњој школи и нарочито у наставку школовања. Ово је важно напоменути, јер се у истом разреду упознаје једна од најважнијих теорема геометрије – Питагорина теорема – која ће у доцнијим разредима скупа са поменутим степеном и полиномијалним рачуном створити основу за упознавање са тригонометријом. Слиједи потпун списак оперативних задатака везаних за планиметрију у седмом разреду:

- Питагорина теорема и њена примјена код свих изучаваних геометријских фигура у којима се може уочити правоугли троугао
- Најважнија својства многоугла и круга

- Приближна конструкција ма којег правилног многоугла и геометријска конструкција појединих правилних многоуглова (са 3, 4, 6, 8 и 12 страница)
- Најважнији обрасци у вези са многоуглом и кругом који се могу примијенити у одговарајућим задацима
- Појам размјере дужи и својства пропорције
- Појам сличности троуглова и њена примјена у једноставнијим случајевима

Задаци математике у осмом разреду

Иако смо се унапријед ограничили на планиметрију, која се изучава од петог до седмог разреда основне школе, овдје желим дати једну примједбу за наставу математике у осмом разреду. Наиме, у овом разреду геометрија се почиње протезати до тродимензионалног простора и тродимензионалних фигура (тачка, права и раван у простору, затим призма, пирамида, ваљак, купа, лопта), те представља посебан изазов за израду софтвера, изнад или паралелно с проблематиком какву имамо у области планиметрије. Важно је напоменути, међутим, да се у свакодневној школској настави и ова област покрива користећи дводимензионална наставна средства, попут свеске и табле, при чему се ослања на ефекте перспективе и пројекције коју ђаци прилично лако препознају захваљујући урођеним способностима. На тај начин, иако пројекат који ћемо развити не покрива тродимензионалне просторе и фигуре, у њему ће свакако бити могуће симулирати их одговарајућим дводимензионалним фигурама. Тако, на примјер, лопта ће моћи бити приказана помоћу једног круга и једне елипсе, ваљак уз помоћ двије дужи и двије елипсе, призма уз помоћ троуглова и/или дужи итд. (Слика 1.)



Слика 1. Илустрација коришћења равних геометријских фигура у циљу симулирања тродимензионалних објеката

Захтјеви који се постављају пред софтверКонцепција рјешења

Допунска и додатна настава у класичном смислу уобичајено се ~~спроводи~~ спроводе „лицем у лице“, било тако што наставник у склопу наставног програма или добровољно осигурава ту наставу за оне којима је потребна, било тако што за допунску наставу родитељи/старатељи ученика плаћају приватне часове стручним лицима. И у једном и у другом случају отежавајуће околности су простор и вријеме. Наставник на овај начин мора лично с ученицима провести пуно вријеме које је потребно за савлађивање

дотичног градива, и за сваког појединца или групу мора спровести задатак испочетка, поново лично учествујући. Ученици, с друге стране, добијају прилику за ову наставу у одређеним временским периодима и у фиксном трајању од једног или два школска часа, те морају искористити прилику да стекну одређено знање чак и у тренуцима када за учење нису расположени или сконцентрисани, из било ког разлога.

Проблеми обију страна могу се превазићи уз помоћ рачунарског софтвера. С једне стране, наставник може користити одређени софтвер да осмисли и опише одређени математички поступак, сними га и потом препусти ученицима на коришћење. С друге стране, ученици овакав производ (како софтвер, тако и поступак који је у њему приказао наставник) могу користити у произвољном тренутку, онда када су расположени и бесплатно, или у најмању руку по знатно мањој цијени од оне коју би родитељи добили за приватне часове. Основни предуслов за овакво спровођење допунске наставе је посједовање рачунара, што, како смо утврдили у првом поглављу овог рада, задовољава нешто преко половине укупног броја домаћинстава у Србији и тај проценат се постепено повећава.

У контексту планиметрије и Плана и програма за основну школу, софтвер који би се користио мора задовољавати сљедеће захтјеве:

Захтјев 1. Лако и прецизно цртање основних равних геометријских фигура: тачка, дуж, троугао, правоугаоник, произвољан многоугао, затим круг, односно кружница, угао и, за потребе обиљежавања и коментара, текст.

Захтјев 2. Промјена већ нацртаних објеката: промјена параметара (трансформација), помјерање (транслација) и ротирање.

Захтјев 3. Памћење појединачних корака одређеног поступка, у оном редослиједу којим се постиже рјешење одређеног задатка, да би се исти доцније могли репродуковати пред очима ученика.

Захтјев 4. Дио пројекта у којем наставник описује жељени поступак мора бити једноставан, не само зато што је потребно да се наставник у њему лако снађе, него и зато што би било добро да наставник може препустити ученику да сам дефинише одређени поступак, било за вјежбу, било за домаћи рад, чиме ученик излази из искључиво пасивне улоге посматрача и преузима иницијативу за сопствено рјешење.

Захтјев 5. За ученике мора постојати и транспарентни дио пројекта, прегледач (енг. *player*), у којем ће моћи посматрати резултат рада наставника и усвојити потребно знање. Ово је битно да би се ученици у одговарајућим тренуцима могли усредсредити само на дотични задатак, не улазећи у појединости самог програма за цртање и његовог окружења.

Захтјев 6. Како су одређени поступци или појединачни кораци поступка такве природе да захтијевају вербално објашњење наставника, пројекту је и поред свих наведених тачака ипак потребна компонента снимања гласа, као незамјењив фактор преношења знања. Наставник треба бити у стању да дефинише тачан тренутак у којем ће се његов гласовни коментар репродуковати у току презентације.

Остваривањем пројекта који посједује поменуте особине омогућило би циљним ученичким групама да у тренутку кад им највише одговара, ван или током наставе, надокнаде пропуштене лекције или стекну допунска знања, са скоро свим елементима које добијају у настави уживо: цртеж је јасан и прецизан, поступак се спроводи у правом редослиједу и у кључним тренуцима чује се гласовно објашњење наставника.

Уколико то прегледач буде дозвољавао, ученик ће имати и једну могућност коју нема ни у једном облику директне комуникације са наставником: да се позиционира на произвољан тренутак у току поступка, врати се неколико секунди или минута уназад ако му нешто није било јасно или жели да утврди знање, или прескочи дио поступка с којим је већ упознат у циљу уштеде времена.

~~У наредном поглављу упознаћемо се с једним софтверским рјешењем које већ постоји у овом циљу, које је слободно и доступно на интернету, али које не пуди, као што ћемо видјети, баш све могућности које смо себи зацртали изнад – „Геогебра“.~~

Постојећа рјешења ~~— „Геогебра“~~

Претрага на Гуглу „интеративни геометријски софтвер“ као први резултат даје чланак на Википедији на енглеском језику, под насловом „Списак интерактивног геометријског софтвера“[1]. Из чланка се види да постоји богатство програма који нуде геометријску интеракцију из различитих перспектива. Неки од њих већ се представљају као едукативни софтвер, а неки се могу посматрати тако иако се представљају као игре или од друге намјене. Остали резултати на Гуглу мимо Википедије представљају мање-више исте резултате. На располагању су „Cinderella“, „Dr. Geo“, „Geogebra“, „KSEG“, „C.a.R.“ и многи други.

Претрагом по резултатима на српском језику, поново се наилази на Геогебру, као и „MiniMath“, који описују Мирослав Марић, Милена Марић и Катарина Радаковић, са Универзитета у Београду и такође скрећу пажњу на Геогебру. Већина доступних програма такође су уско дефинисане, тако да представљају геометријске односе уз прецизно дефинисану интеракцију. Геогебра се издваја обухватношћу различитих алата и погледа, на одређени начин обухватајући многе од преосталих програма. Такође се издваја обимом прихваћености у настави и ван наставе, улазећи чак и академски контекст. У наставку овог поглавља посветићемо пажњу овом софтверу, да бисмо изучили до које мјере би могао да задовољава захтјеве које смо поставили у овом раду и шта можемо научити из њиховог примјера.

Formatted: Bullets and Numbering

Основни подаци о програму „Геогебра“

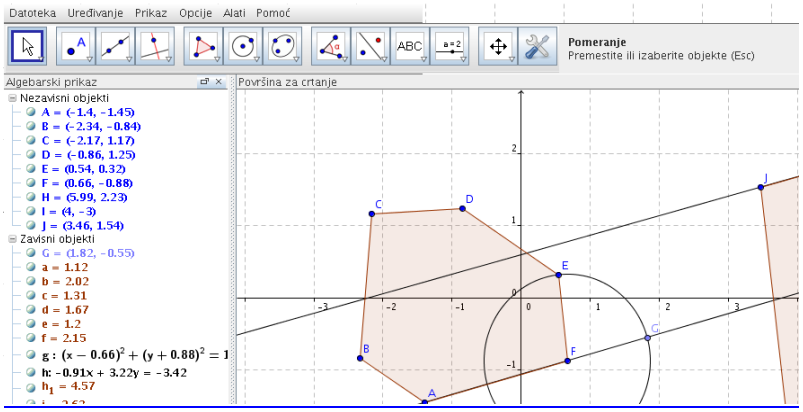
Геогебра (енг. *GeoGebra*) је интерактивна апликација за геометрију, алгебру и анализу, намијењена ученицима и наставницима. Пројекат је настао као дјело једног аутора, Маркуса Хохенвартера, на Универзитету у Салцбургу, 2001. године. Од њеног настанка до данас, међутим, већи број програмера-волонтера учествовао је у развоју апликације, а највећи дио пројекта је потпуно слободан, односно под лиценцом слободног кода. Преводиоци-волонтери утицали су да пројекат буде преведен на преко четрдесетидевет свјетских језика (мај 2012), укључујући српски у латиничном писму. Писан је користећи програмски језик „~~јава~~Јава“, што имплицира да је доступан за различите оперативне системе и хардверске платформе. Имајући у виду све речено, Геогебра је потпуно бесплатна и широко доступна за коришћење.

Геогебра је популарна на свјетском нивоу, нарочито у оквиру наставе математике у основним и средњим школама. У љето 2009. године одржана је прва међународна конференција под називом „Прва међународна конференција о Геогебри 2009“ (енг. *First International GeoGebra Conference 2009*) са темом „Примјена Геогебре у настави математике“ (GeoGebra Conference Report, 2010). У Србији, професори Ђорђе Херцег и Драгослав Херцег са Универзитета у Новом Саду и професор Мирослав Марић са Универзитета у Београду раде на популаризацији овог софтвера у земљи (Марић, 2011; Херцег, 2007).

[1] List of interactive geometry software, http://en.wikipedia.org/wiki/Interactive_geometry_software

Основне особине и начин коришћења

Геогебра нуди иконичко и симболичко представљање математичких објеката паралелно у геометријском и алгебарском прозору (Херцег, 2007). Ово значи да Геогебру одликује дуалност, тј. да је главни прозор подијељен на два потпрозора: геометријски, који се често назива и простор за цртање, и алгебарски, односно симболички потпрозор (Слика 2). Дуалност се огледа у томе да у сваком тренутку стање у једном потпрозору одговара стању у другом потпрозору; промјена у једном изазива промјену у другом, и обратно.



Слика 2. Главни прозор Геогебре, са алгебарским приказом (лијево) и главним, геометријским приказом (десно)

Руковање геометријским потпрозором обавља се мишем и тастатуром. Нови објекти се цртају користећи низ алатки које су на располагању у палети алатки. Постојећи објекти на сцени су интерактивни: могу се одабрати (селектовати), трансформисати и ротирати; наравно, може се промијенити њихов положај. Десним кликом миша на произвољан објекат добија се контекстуални мени који пружа додатну функционалност.

Алгебарски прозор нуди нам могућност прецизног одређивања параметара компоненти које смо дефинисали уз помоћ миша и тастатуре у геометријском прозору, али у случају да је потез миша тај који је мјеродаван, овај прозор нам може послужити за читање параметара које смо дефинисали.

Геогебра подржава различите јединице мјере, системе и конвенције – углови се могу дефинисати у степенима или радијанима; може се користити и Декартов и поларни координатни систем; објекти се могу задавати преко дословних координата или путем једначине, односно формуле. На жалост, изузетак је децимални зарез: мора бити коришћен англосаксонски зарез, односно децимална тачка.

Рад са објектима

У оквиру Геогебре могу се дефинисати тачке, праве, полуправе, вектори, дужи, конусни пресјеци, као и графици функција. Сам програм је у стању да рачуна изводе и интеграле задатих функција.

Тачка се образује најприје одабиром алатке *тачка*, а потом кликом миша на произвољно мјесто у геометријском прозору. Одабиром опције *тачка на објекту,* а затим кликом на постојећи објекат, образује се тачка која припада том објекту. Одабиром опције *пресјек два објекта*, пружа нам се могућност да кликнемо на два постојећа објекта, један за другим, након чега се образују нове тачке које се налазе на пресјеку ових објеката.

Дуж се може образовати задавањем двију тачака, користећи опцију *дуж између двије тачке*. Дуж се може образовати и на основу једне тачке, задавањем дужине дужи, која је за почетак водоравна, а њена друга тачка се може помјерати.

Права и **полуправа** се формирају као и дуж. Одабира се опција *права кроз двије тачке* или *полуправа кроз двије тачке*, а затим се одабирају двије одређујуће тачке.

Вектор се најједноставније одређује двјема тачкама, али постоји и могућност дефинисања вектора на основу једне тачке и постојећег вектора. Нови вектор ће тада имати интензитет и правац као постојећи вектор, али ће његов почетак бити у задатој тачки.

Кругу, **кружном луку** и **кружном одсјечку** је посвећена нарочита пажња у палети алатки. Круг је могуће задати центром и једном тачком, центром и полупречником, трима тачкама које му припадају и трима тачкама од којих двије представљају дуж која дефинише полупречник, а трећа положај центра круга. Кружни лук се може одредити центром и двјема тачкама, трима тачкама, или као полукружница одређена двјема наспрамним тачкама. Напослијетку, кружни одсјечак се може одредити као и кружни лук, с том занимљивом разликом да не постоји опција дефинисања полукружног одсјечка помоћу двију наспрамних тачака.

Конусни пресјеци **елипса**, **хипербола** и **парабола** имају по један начин дефинисања, редом: двјема жижама (фокусима) и тачком која припада елипси, двјема жижама (фокусима) и тачком која припада хиперболи и, за параболу, жижом и директрисом. Занимљива опција је дефинисање произвољног конусног пресјека помоћу пет тачака: прве четири тачке ће бити само приказане, док пета тачка одређује посљедњу димензију и тип конусног пресјека – илустративно је посматрати како помјерањем пете тачке конусни пресјек може промијенити тип, па тако из елипсе настаје парабола, а даљим помјерањем хипербола чији други дио као да „излази из бесконачности“ у коју је отишла друга страна првобитне елипсе.

Геогебра обилује корисним алатима за геометријске конструкције, поред наведених елементарних врста објеката:

- Права нормална на задату праву или паралелна њој
- Симетрала дужи
- Тангента на задати конусни пресјек, круг или другог типа
- Осна и централна симетрија објекта
- Мјерење површине, растојања и нагиба
- Ротација објекта око тачке за неки угао итд.

За руковање сценом овдје су присутне уобичајене алатке за помјерање сцене, увеличавање и умањивање (*зумирање*), убацивање постојеће слике (*битмапе*), привремено сакривање објекта итд.

Зависни и независни објекти

Једна од најважнијих особина Геогебре је постојање тзв. *зависних* и *независних* објеката. Независни објекти су они чији су сви параметри задати директно, координатама. Зависни објекти су они чији је бар један параметар задат користећи својство неког другог, постојећег објекта. На примјер, тачка која је настала кликом миша на празној површини је независан објекат, исто као и круг задат трима независним тачкама. С друге стране, ако је једна од одређујућих тачака круга припадала неком другом објекту, тај круг је зависан од те тачке, односно објекта којем припада. Вриједност овакве концепције крије се у сљедећем: уколико накнадно промијенимо неки параметар објекта од којег наш објекат зависи, и наш објекат ће се промијенити у складу с тим. На овај начин могуће је поставити неке параметре објекта као зависне, а затим посматрати ефекте те зависности при промјени првобитног објекта. Зависности је могуће и ланчано везати, тако да објекат Б зависи од објекта А, објекат В од објекта Б, објекат Г од објекта В итд.

Анимација

Геогебра има скромну подршку за анимацију. Она се уводи коришћењем алата под називом „клизач“. Клизач је апстрактни објекат који заправо представља бројевни распон, а могуће је дефинисати његов почетак, крај и корак. Клизачу се даје произвољно име, а то име потом дјелује као независна промјењива. Други објекти могу користити ту промјењиву као један од својих параметара, те постају зависни од тренутног положаја на клизачу. Промјеном положаја на клизачу аутоматски се мијења објекат који зависи од њега.

Анимација објекта уз помоћ клизача постиже се тако што се неки параметар дотичног објекта веже за клизач, а потом се кликне десним тастером миша на клизач и из контекстуалног менија одабере опција „анимирај“. Положај клизача почиње аутоматски да се мијења, а самим тим и дотични објекат.

Постоји и опција ручно вођене анимације, тако што се одабере угао, број или зависна тачка, а затим притиском тастера + или – на тастатури ручно модификује ова вриједност.[2]

Други корисни алати

За крај овог поглавља, поменућемо још неке важне особине пројекта у цјелини:

- Постоји опција просте анализе односа двају објеката. Анализа се обично врши нумерички, па нпр. за двије одабране дужи програм може извијестити да нису једнаке, али су једнаке дужине.
- Из главног менија може се одабрати језик приказа. Промјена ступа на снагу истог тренутка и, бар у случају српског језика и по личној провјери аутора овог рада, са исправно преведеном математичком терминологијом.
- Постоји засебан „калкулатор вјероватноће“, који се отвара у засебном дијалогу.

2 Детаљне инструкције како произвести анимацију могу се погледати на страници на енглеском језику, на веб адреси http://wiki.geogebra.org/en/Animation

Могуће је одабрати једну од четрнаест понуђених типова дистрибуције вјероватноће (нормална, експоненцијална, Кошијева, Студентова, Поасонова итд.), да ли је расподјела лијева, десна или двострана и сличне параметре. При промјени сваког параметра у горњем дијелу дијалога се ажурира или приказује график расподјеле.

- Међу осталим алаткама које смо поменули, постоји и алатка *оловка*, која омогућава цртање произвољне шаре на сцени.
- Сцена има неколико режима рада: основни (геометријски и алгебарски приказ), прости (само геометријски, за ниже разреде основне школе) и основни са табелом (за потребе рачунања или вођења евиденције обезбијеђена је табела у стилу оне коју познајемо у комерцијалним програмима попут „ексела“.

Излазни формати

Математичка конструкција коју образујемо користећи софтверски пакет Геогебра може се снимити у нативном формату Геогебре, са датотечном екстензијом .ggb. Овај формат омогућава каснији наставак рада на конструкцији, са свим алатима које Геогебра иначе пружа. Уколико аутор конструкције жели објавити рад да би био доступан другима, чак и онима који немају приступ Геогебри или не познају рад у њој, на располагању је неколико стандардно подржаних формата:

- Површина за цртање као слика; подржани су формати .png (portable network graphics), .pdf (portable document format), .eps (encapsulated postscript), .svg (scalable vector graphics) и .emf (enhanced metafile)
- Динамички цртеж као веб страница; подржан је стандардни веб формат HTML, у оквиру којег се може аутоматски генерисати јава аплет. Овај аплет скупа са веб страницом може се снимити на локалном рачунару, а затим отпремити на јавни сервер по жељи, одакле га могу преузимати заинтересовани корисници. Занимљива опција је и *Upload to GeoGebra Tube*, која омогућава да се произведени рад бесплатно отпреми на званични сајт Геогебре за радове својих корисника, одакле га такође може преузети сваки заинтересовани корисник. Ова опција нарочито је корисна за наставнике који немају сопствени веб-сервер.

Поред наведених формата, који су најчешће у употреби, конструкцију је могуће извозити и као анимирани GIF формат, као и генерисати одговарајући код за PSTricks (скуп макроа који омогућавају увожење PostScript цртежа у Tex или Латех), Asymptote (дескриптивни језик за векторску графику, који омогућава окружење за техничко цртање базирано на координатама тачака) и PGF/Ti*k*Z (језик за произвођење векторске графике на основу геометријског, односно алгебарског описа).

На веб адреси www.geogebratube.org може се прегледати пространан скуп јавно доступних конструкција снимљених у Геогебри, категорисаних по датуму, области математике, популарности, итд.

Закључак и анализа

Геогебра је без сумње врло користан алат за наставнике математике, али и за наставнике других предмета у којима се интензивно користи математика (у првом реду физика). Највећа корист пројекта Геогебра, по мишљењу аутора овог рада, лежи у концепту зависних и независних објеката; резултат рада тако може да се понуди

ученицима, који сопственим мијењањем параметара могу проучавати оно што је и у Плану и програму дефинисано као један од основних задатака, а то је „да се ученици упознају са најважнијим равним геометријским фигурама и њиховим узајамним односима“.

Могућности Геогебре за описивање цјелокупног математичког поступка корак по корак, међутим, скромне су. Клизач, као метода анимације, подесан је за описивање једног корака, евентуално у садејству са другим објектима, али у принципу служи такође за приказивање међусобних односа објеката, или какве резултате производи промјена одређеног параметра објекта у одређеном распону. Међутим, ова врста анимације не пружа могућност описивања сложеног математичког поступка за који су потребне минуте или десетине минута.

Погледајмо детаљно како и у којој мјери Геогебра задовољава захтјеве 1-6 које смо поставили у поглављу „Концепција рјешења“, изнад.

Захтјев 1. Геогебра је изузетно удобна за коришћење и има пријатан кориснички интерфејс. Све расположиве врсте објеката се могу добити кликом на дугме миша, као и нумеричким уносом. Програм подржава цртање свих врста објеката који су потребни за наставу планиметрије од петог до седмог разреда. Захтјев задовољен: **да.**

Захтјев 2. Сваки објекат дозвољава промјену својих параметара, бројно или уз помоћ миша, па према томе закључујемо да је трансформација подржана. Постоји алатка „транслација за вектор“, помоћу које можемо помјерити објекат, као и „ротација тачке за угао“, помоћу које можемо ротирати објекат. Међутим, и транслацијом и ротацијом заправо добијамо *додатни* објекат, који представља слику операције коју смо извршили, па након дотичне операције видимо два објекта на екрану: објекат у првобитном стању и објекат који представља слику функције. Такође, слика објекта остаје зависна од првобитног објекта и не може се мијењати, што нас спречава да ланчано трансформишемо објекат и производимо геометријски поступак у правом смислу те ријечи. Захтјев задовољен: **дјелимично.**

Захтјев 3. Као што смо објаснили на почетку овог поглавља, у пасусу о анимацији, овај захтјев **није задовољен.**

Захтјев 4. Једноставност корисничког интерфејса Геогебре, у контексту услова да треба бити довољно једноставан да и ученици (дјеца) могу обављати задатке у њему, јесте дискутабилна тема. Свакако да би требало да ову особину процијени стручњак из области корисничког искуства (енг. *user experience*), односно дизајна, али проблематично је већ само постојање математичких алата који превазилазе градиво за основну школу (нпр. конусни пресјек са пет тачака, график функције, калкулатор вјероватноће, полара, фитована права, итд), јер могу збунити ученика, отежати му да нађе алате који су њему потребни, створити му психолошки отпор да је програм превише сложен, да је математика тешка итд. Захтјев задовољен: **можда.**

Захтјев 5. Геогебра нема стварни „плејер“, тј. прегледач, као засебну компоненту пројекта, али пружајући подршку извоза документа у интерактивну веб страницу она може одвојити ученика од процеса израде документа и пружити му могућност да се усредсреди само на математички поступак који је у питању. ~~Услов~~ Захтјев задовољен: **да.**

Захтјев 6. Геогебра нема подршку за звук. Звук не може бити ни увезен, ни снимљен. Ово је, по мишљењу аутора овог рада, природна посљедица саме концепције

Геогебре у којој анимација није подржана у контексту ~~наших~~ потреба које смо описали, те се не може узети као замјерка самом програму, али истовремено онемогућава оне операције због којих нам је звук потребан: додатно објашњење дато природном ријечју, на језику слу-шаоца и гласом човјека. ~~Услов~~ Захтјев задовољен: **не.**

Резултати ове анализе приказани су у табели 1.

Захтјев	**Задовољен у Геогебри**
Лако и прецизно цртање потребних равних фигура	Да
Промјена објеката, трансформација, ротација, транслација	Дјелимично
Могуће приказати цјелокупан поступак у виду анимације	Не
Једноставност употребе да га и дјеца могу користити	Можда
Могуће изоловати се само на поступак о којем се говори	Да
Подршка за гласовне коментаре, односно подршка за звук	Не

Табела 1. Преглед задовољености захтјева за софтвер подршке настави планиметрије за основну школу

Са два „не“ и једним „дјелимично“, од укупно шест најважнијих ~~услова~~захтјева, закључујемо да је у нашем конкретном случају, гдје говоримо о допунској и додатној настави за област планиметрије у трајању од три школске године, потребно обезбиједити посебан софтвер који би задовољио све што је наведено. Ученицима је потребан опис цјелокупног поступка помоћу којег се, на примјер, конструише круг уписан у троугао, полазећи од датог троугла и користећи само шестар и лењир, а да при том имају увид у појединачне кораке помоћу којих се долази до овог резултата. У програмском пакету Геогебра ученици су, такође, присиљени да сами извлаче закључак зашто је извршен одређени корак, а бројни су примјери када је потребно унапријед објаснити шта ће се и како обавити у остатку поступка. Ово је могуће учинити уз помоћ текста, али уз подршку за звук ученику је објашњење природније и више скреће пажњу. На послијетку, уз помоћ гласовног коментара могуће је пружити и општу представу шта *намјеравамо* урадити у остатку поступка, обухватајући тиме неколицину корака који слиједе.

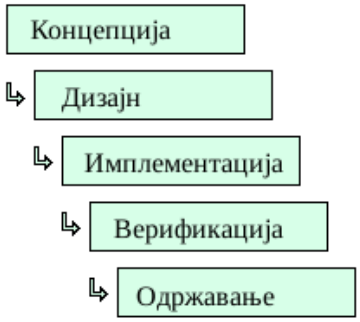
Геогебру оцјењујем као подесну и пожељну за ученике старијих разреда основне школе који су већ заинтересовани за математику, као и за ученике средњих школа и факултета, такође довољно заинтересоване да лако превазиђу учење радног окружења Геогебре и способне да сами анализирају повезаности и зависности геометријских и алгебарских објеката. Оцјењујем је као мање подесну за ученике млађих разреда основне школе који тек треба да стекну основна знања из области планиметрије и као потпуно неподесну за ученике истих тих разреда који из одређених разлога већ имају проблема са савлађивањем градива.

Дизајн и имплементација софтвера

„Лењир и шестар“ — ~~имплементација рјешења~~

Методологија израде

При изради софтверског пројекта неопходно је одабрати одговарајућу методологију развоја. Из разумљивог разлога што овај мастер рад треба да имплементира једна особа, у самом почетку морамо одбацити све методологије које обавезно подразумијевају тимски рад, од којих су ~~најпознатијинајпознатије~~: ~~еџајл~~ агилна (енг. *Agile*), скрам (енг. *Scrum*) и екстремно програмирање. Можемо такође одбацити методологије којима је циљ смањење потрошње времена и новца, попут ~~методологија~~ „~~лин~~мршаве“ (енг. *lean*) методологије ~~(енг. *lean*)~~ и РАД (енг. *RAD, rapid application development*) методологије. Умјесто наведених критеријума, с обзиром да говоримо о пројекту *одређене намјене*, односно с обзиром да су нам потребе корисника унапријед познате, и пошто је у питању академски рад, можемо изабрати неку од „традиционалнијих“ методологија рада, које нам дозвољавају правилан дизајн и које не захтијевају вођење бриге о сталним промјенама захтјева клијента, као што се дешава у случају израде комерцијалног софтвера. Од таквих метода издвајамо В-модел (енг. *V-model*) и „водопад“ (енг. *waterfall*). В-модел се заправо заснива на водопаду, с том разликом да се уважава аналогија између појединих фаза развоја и тестирања, на примјер између фазе развоја „захтјеви и архитектура“ и фазе тестирања „системско тестирање и валидација“. Иако дозвољава правилан дизајн и практично линеаран развој, овај систем је осмишљен такође као надоградња водопада за окружења гдје постоје засебни тимови који паралелно раде и комуницирају међусобно. За најједноставнији приступ и најчистији дизајн изабраћемо, сходно овој расправи, прост систем „водопада“ (слика 3).



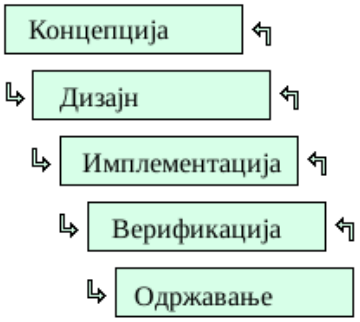
Слика 3. Модел „водопад“ развоја софтверског пројекта: пројекат пролази секвенцијално кроз пет фаза

Модел **водопада** представља секвенцијални дизајн у којем се напредак пројекта, док пролази кроз различите фазе развоја, приказује у строго силазној путањи, као

водопад. Ове фазе су (Royce, 1970):

1. Концепција; цјелокупан опис функционалности које пројекат треба да омогући и како ће изгледати кориснички интерфејс
2. Дизајн; на основу потреба наведених у претходној фази долази се до општег дизајна пројекта у смислу архитектуре софтверских компонената
3. Имплементација; потребно је уложити конкретан труд на изради појединачних компонената дизајна
4. Провјера; по свршетку развоја, провјерава се функционална исправност засебних компонената система и пројекта у цјелини
5. Одржавање; потребно је мотрити на функционалност система у пракси, излазити у сусрет корисницима, у смислу проблема на које наиђу и додатних захтјева

У пракси овај систем наилази на проблеме. Наиме, у идеалном случају требало би да сваки појединачни корак буде извршен тако да нема потребе за његовом корекцијом након преласка у сљедећу фазу. У реалности ниједан корак не може бити доведен до тог нивоа савршености, те у зависности од тога колико је био близак том нивоу или далеко од њега, биће потребно кориговати претходне кораке тако да се омогући успјешно комплетирање пројекта и да он буде занемарљиво другачији од првобитне концепције.



Слика ~~34~~ 34. Када модел „водопад“ крене наопако

Зато при процесу израде у свакој фази треба имати на уму да ће и мањи недостаци у тој фази проузроковати веће проблеме у наредним. Рано откривање грешке током дизајна одузима мање времена и труда него откривање појединих грешака у имплементацији које су настале због те грешке у дизајну. Исто тако, мање времена и труда потребно је да би се написала исправна функција у фази имплементације, него накнадно тражење грешке, најприје у току провјере функционалности, а потом и у оквиру саме имплементације, те је накнадно исправљати.

У остатку рада ~~трудићемо се да останемо~~ остаће се у оквирима изабране методологије рада, тако што ће~~мо~~ се кроз поглавља пратити поједине фазе развоја наведене у тачкама изнад.

Концепција

Срж концепције већ ~~смо представили~~ је представљена у Уводу овог рада, у поглављу „Концепција рјешења“. Идеја се обично рађа из одређене потребе, а до

потребе смо дошли кратком анализом текућег стања образовања по питању усвојености модерних наставних средстава, успјешности ученика у области математике и спремности наставног особља да усваја нове методе. Како је анализа показала потребу за пројектом који даје одговор на захтјеве 1-6 из поменутог поглавља, овдје ћемо наставити детаљну разраду једног таквог система.

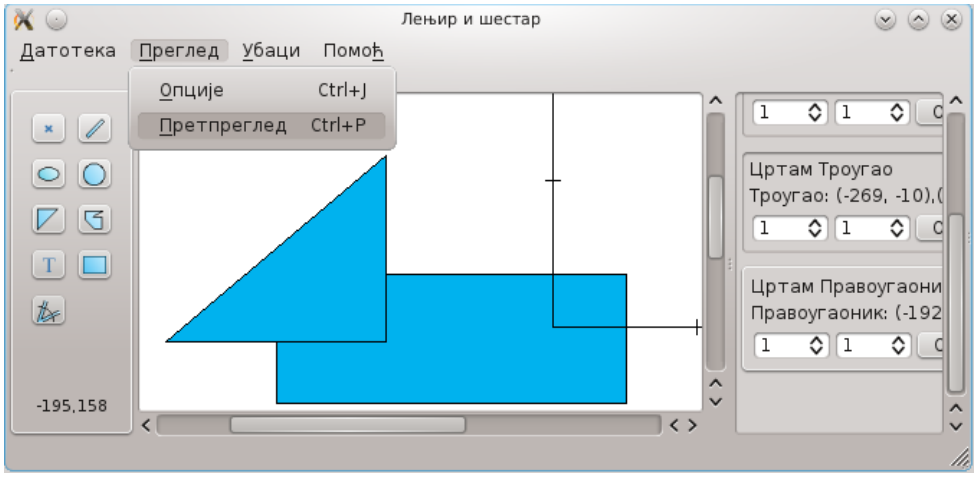
За радно окружење у којем се може документовати основна концепција и њени детаљи аутор овог рада је одабрао окружење „вики“ (енг. *wiki*). Вики је у својој суштини веб сајт, али дизајниран на такав начин да корисници могу мијењати његов садржај. Његова првобитна примјена била је у развоју данас највеће слободне мрежне енциклопедије на свијету, Википедије. Поред ове примјене, вики софтвер се користи у разним ситуацијама у којима је потребна интеракција и сарадња у циљу развоја одређеног пројекта, од писања званичних упутстава и документације, до писања личног дневника у циљу боље организације података (Britannica, 2007).

У прилогу представљамо базу података потребну за приказ концепције у вики формату, а овдје ћемо је приказати у скраћеним цртама.

Радни простор

Највећи дио главног прозора **апликације за цртање** заузимаће простор за цртање, односно главни простор. Он ће се растезати и скупљати у складу са величином корисничког прозора тако да увијек заузима највећи његов дио, по водоравном и усправном правцу. Лијево од простора за цртање биће узак стуб са по једним дугметом за цртање сваког типа објекта. Десно од простора за цртање постојаће потпрозор „историја анимација“ (v.i.). Призор из радног простора може се видјети на слици 5.

Изнад простора за цртање постојаће уска трака коју ће различити видови формирања анимације користити као контекстуалну палету параметара: док се буде цртао објекат, ту ће бити представљене графичке контроле за нумерички и текстуални унос параметара тог објекта.



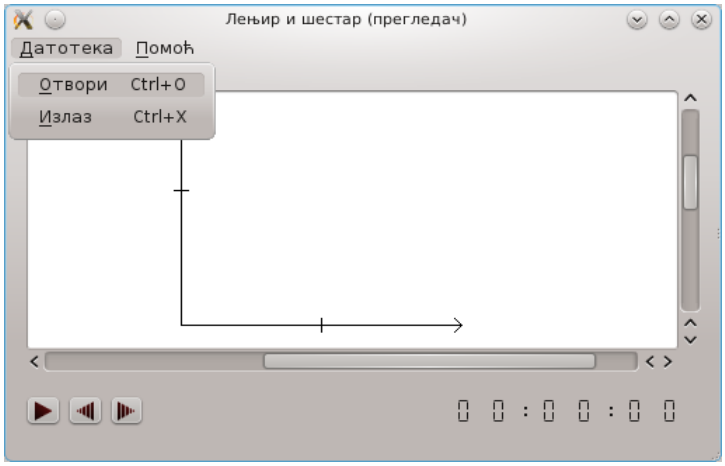
Слика 5. Главни прозор: главни мени у врху, доле алатке за цртање, потом десно главни простор и крајње десно историја анимација.

Главни мени садржаће команде за:

- Рад са датотеком (снимање, отварање, промјена имена)
- Приступ прозорима за: опције, снимање звука, основне информације о програму
- ~~Претпреглед~~ Прелиминарни преглед документа, команда која покреће екстерни прегледач

У простору за цртање корисник ће за сваки објекат на сцени имати на располагању одговарајући контекстуални мени, који ће зависити од тренутног стања сцене: ако је у току цртање новог објекта, контекстуални мени ће се односити на акције потребне за његову конструкцију, а уколико је и посљедњи објекат на сцени „комплетиран“ (v.i)

У прозору **прегледача** биће изузети палета за цртање, историја анимација и контекстуална палета, као и већи дио команди у главном менију: за рад са датотеком биће задржане само команде за отварање документа и излаз из апликације, а од преосталих команди биће задржана само она за приступ прозору са основним информацијама о програму/прегледачу. На дну апликације прегледача биће дугмад за пуштање снимка, паузирање, убрзано помјерање неколико секунди унапријед и унатраг, те графички приказ протеклог времена од почетка снимка (слика 6).



Слика 6. Прегледач, главни прозор са менијем и командама за руковање снимком

Спецификација објеката за цртање

Постоји укупно осам врста објеката за цртање: тачка, дуж, троугао, правоугаоник, елипса, многоугао и текст. Сваки од њих има себи својствене улазне параметре и помоћна средства, али све врсте објеката имају одређена заједничка својства. Ова заједничка својства су:

1. Сваки објекат има име (текст) и боју.
2. При цртању, за сваки објекат ће бити доступна тастатурна пречица „контрол“; ако се држи притиснутим током дефинисања било које тачке објекта, дефинисана тачка ће „скакати“ до најближе тачке најближег сусједног објекта, ако је та тачка на удаљености мањој од Л пиксела од текуће позиције курсора миша. У литератури на енглеском језику ова могућност се уобичајено зове „пуцкање (прстима)“, енг. *snap, snapping*.

Улазни параметри за различите врсте објеката, поред имена и боје који су заједнички за све врсте објеката, биће сљедећи:

- Тачка: тачка (тачка)
- Дуж: почетак и крај (тачке)
- Троугао: три тачке
- Правоугаоник: горњи лијеви угао (тачка), дужина и висина (реални бројеви)
- Кружница: центар (тачка) и полупречник (реалан број)
- Елипса: центар (тачка), два полупречника (реални бројеви)
- Многоугао: тачке
- Текст: садржај (текст), положај горе лијево (тачка), име фонта (текст), величина (један од бројева из скупа {8, 10, 12, 14, 16, 18, 22, 26, 32}) и стил фонта: подебљано, искошено, подвучено.

При цртању неког објекта, сваки од параметара дотичног објекта биће дефинисан једним кликом миша. На примјер, да би се нацртао правоугаоник биће потребно кликнути једном на тачку гдје желимо да буде његов горњи лијеви угао, још једном на тачку која је по хоризонталној оси на оној удаљенести од прве тачке која представља његову дужину и још једном за дефинисање висине правоугаоника. За потребе текста треба подржати тастатурни унос свих слова српске азбуке и у ту сврху користити кодни распоред „јуникод“ (енг. *Unicode*).

Анимације

Објекти на сцени представљају основу анимације, а анимације се користе да би се приказао одређени геометријски поступак. Све анимације имају вријеме трајања у секундама, које ће диктирати брзину исцртавања при репродукцији.

- **Исцртавање**; Основна врста анимације представља само исцртавање објекта. Дефинишући нови објекат на сцени, корисник имплицитно додаје и нову анимацију у процесу описивања одговарајућег математичког поступка. Сваки објекат ће се при репродукцији поступка исцртати постепено, тачку по тачку, оном брзином коју је корисник дефинисао.
- **Промјена параметара објекта**; Потребно је обезбиједити постепену промјену дотичних параметара или дотичног параметра из старог стања у ново, у оном временском року који корисник одреди за трајање дотичне анимације. Овој могућности се приступа десним кликом миша на одабрани објекат (контекстуални мени) и бирањем команде „трансформација“.
- **Ротација**; Анимација памти број степени и преко 360°, да би се објекат евентуално ротирао и више пута око своје Z-осе. Такође, иницијални центар ротације је центар објекта, онако како га дефинише дотични објекат, али могуће га је помјерити (мишем) и затим извршити ротацију око те тачке. Добија се из контекстуалног менија жељеног објекта.
- **Помјерање**; Потребно је обезбиједити да се објекат континуално помјери из почетног у крајњи положај. Бира се такође из контекстуалног менија.

Одабиром неке од команди за додавање анимације (цртање новог објекта или трансформација постојећег) корисник улази у *режим дефинисања* анимације. То значи да су му изнад простора за цртање доступне графичке контроле за прецизно дефинисање анимације и да се кликом на десни тастер миша добија контекстуални мени који се односи на анимацију која је у току. Корисник остаје у овом режиму све док не притисне дугме „завршено“, крајње десно у анимацијској палети. Тада се враћа у *режим контролисања* постојећих објеката, у којем је могуће одабрати други објекат и измијенити га, или нацртати нови.

Све овакве измјене – додавање објекта или трансформација постојећег – представљају нову анимацију која се памти у опису документа. Памте се само почетак и крај – прије измјене и послије измјене. Прегледач ће доцније бити способан да репродукује линеарну путању између та два стања, по питању свих параметара.

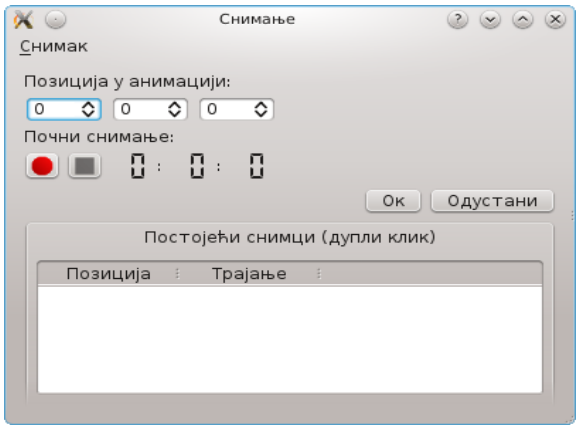
У режиму дефинисања једног корака, у случају анимација ротације, трансформације и транслације, биће доступне ручице за дефинисање трансформације курсором миша. По свршетку дефинисања анимације, ручице ће нестати.

Звук

Програм ће бити снабдјевен могућношћу снимања звука. Звук ће бити могуће само снимити, тј. неће бити подржано увожење произвољне постојеће аудио датотеке, из разлога што подршка за звук постоји у циљу снимања гласовног коментара наставника, а не из естетских разлога.

Звук се може снимити и поставити између двију анимација, прије прве анимације или након посљедње. Програм неће подржавати никакве посебне ефекте и звук ће бити репродукован у цјелини онако како је снимљен, у реалном времену.

Звук ће се додавати кликом на одговарајућу графичку контролу (то може бити команда из главног падајућег менија или дугме на произвољном мјесту). Одабиром ове опције отвориће се посебан графички дијалог (слика 7).



Слика 7. Дијалог за снимање звука. *Позиција у анимацији* означава тренутак када звук који се сними отпочиње с репродукцијом

Историја анимација

Десно од простора за цртање стајаће *историја анимација,* одјељак гдје је приказана листа досадашњих анимација. То обухвата и креирање објеката и друге анимације на њима. Посљедња ставка у листи (која је на врху) имаће у контекстуалном менију ставку "Измени", која ће дозволити да се та посљедња анимација врати у режим дефинисања. На преостале ставке моћи ће се кликнути лијевим тастером миша, чиме ће се узроковати да се ревертују анимације које су дошле послије ње, односно да се сцена врати у стање када је она била посљедња додата анимација. Кликом на ставку неке од ревертованих анимација, на сцени ће се поново извршити ревертоване анимације и

сцена ће се довести у стање када је ова анимација била посљедња додата. Тако ће се омогућити више од традиционалног концепта „врати корак“ (енг. *undo*) и прећи на концепт „историје“ какав имамо код производа „Адоубија“ или „Корела“. Ако корисник одабере да изврши операцију у тренутку кад је сцена враћена на неки од претходних корака, апликација ће га замолити да потврди, јер на овај начин одбацује ревертоване анимације и на њихово мјесто убацује нову. Корисник мора да врати сцену на посљедњи корак да би могао да настави одатле.

Складиштење

Програм омогућава кориснику да сними текући документ и отвори неки постојећи. Датотеке имају екстензију *HWD* (скраћеница за енг. *HomeWork Document*). Потребно је снимити све податке о објектима и анимацијама које су вршене на њима, почетно и крајње стање. У менију „Датотека“ за ове сврхе обезбиједити команде „нови“ (документ), „сними“, „сними под другим именом“ и „отвори“.

Податке снимити у формату *XML*. Овај формат омогућава запис произвољног пакета података у проширивом формату, безбједно за мрежни пренос и омогућава кодни распоред „јуникод“. Звучни запис, у циљу безбједног преноса у оквиру XML документа, биће кодиран помоћу алгоритма *Base64*, чиме ће цио пакет постати текстуалне природе и моћи се отворити уз помоћ једноставног текстуалног едитора. Пошто је XML интуитиван, корисник ће моћи одабрати и да ручно уређује документ или да генерише процес на основу неке формуле и уз помоћ неког једноставнијег генератора текста.

Примјер садржаја датотеке у формату *XML* може се видјети у прилогу 1:

```
<scene>
 <animations>
  <animation duration="1" type="drawing" pause="3">
   <object posx="0" posy="0" type="Triangle" name="Triangle" color="">
    <triangle cx="314" bx="123" cy="-51" ax="-220" by="107" ay="-51"/>
   </object>
  </animation>
  <animation targetName="Triangle" duration="1" type="transformation">
   <mirror posx="0" posy="0" type="Angle" name="Triangle_m" ...>
    <angle beta="58.4" showLines="false" .../>
   </mirror>
  </animation>
 </animations>
 <sounds>
  <sound start="3"><![CDATA[U4BTgFuAW4BegF+BQgGYBBgEKAV...]></sound>
 </sounds>
</scene>
```

Прилог 1. Садржај HWD датотеке: анимације су описане XML структурама, а звук снимљен у Base64 пакете

Дизајн

При изради дизајна морају се издвојити најкрупније функционалне цјелине, а потом их, имајући на уму софтверске технологије која ће се користити, пресликати у ентитете дизајна који се могу даље имплементирати. С друге стране, прије издвајања функционалних цјелина вриједи имати на уму софтверске технологије које ће се користити, због различитих парадигми које те технологије представљају или носе

собом, као и њиховог сопственог дизајна. На тај начин, одговор на једно питање захтијева најприје одговор на друго питање, и обрнуто. Пошто је скуп доступних софтверских технологија ограничен, почећемо с овим питањем.

Софтверске технологије

С обзиром да говоримо о апликацији која служи за руковање геометријским објектима, природно се намеће парадигма **објектно-оријентисаног програмирања**:

1. Сваки геометријски објекат има параметре (атрибути класе)
2. Два геометријска објекта могу међусобно сарађивати (методе класе)
3. Два или више геометријских објеката су у одређеном односу (методе класе)
4. Над различитим врстама геометријских објеката могу се вршити исте операције на различите начине, нпр. ротација, исцртавање, итд. (виртуелне функције)

Овде као општу одредницу наводим и то да сам као оперативни систем за развојно окружење изабрао ~~линукс~~Линукс, верзија Слеквер 13 (енг. Slackware 13), који је слободан, бесплатан, представља снажно развојно окружење због огромног броја програмерских алата који подразу-мијевано долазе уз њега, а подржава и софтверске технологије које смо, како ћемо видјети испод, одабрали за развој пројекта.

Одабир програмског језика

Основно својство програмског језика који изаберемо мора бити могућност преносивости; ово стога што планирамо омогућити приступ апликацији за што већи број ученика, а то на првом мјесту зависи од оперативног система који је инсталиран на њиховом рачунару. Што апликација буде подржавала већи број различитих платформи, то је већа вјероватноћа да ће произвољан ученик бити у могућности да је покрене.

Друго својство о којем морамо повести рачуна јесте ефикасност дотичне технологије. С обзиром да намјеравамо радити с графичким објектима и анимацијом, већ сама логика апликације захтијеваће релативно велики удио у раду процесора и меморије рачунара, те би било добро за програмски језик изабрати онај који пружа максималне перформансе, уз поштовање поменутог принципа преносивости.

На послијетку, како смо већ већ истакли, програмски језик који ћемо користити треба да подржава објектно-оријентисано програмирање.

У табели 2. упоредићемо својства двају популарних програмских језика који нам пружају особине изнад, Јаве и C++-а.

	C++	**Јава**
Преносивост	„Пиши једном, компајлирај свуда“	„Пиши једном, покрећи свуда“
Приступ системским функцијама	Да, али у складу са оперативним системом који се користи, тако да се тиме нарушава преносивост	Не
Објектно оријентисан	Да, уз могућност процедуралног, функционалног и шаблонског псеудо-програмирања (Eckel, 2003)	Строго објективан програмски језик
Пренос параметара	Подржани су показивачи, референце и преношење по вриједности	Елементарни и референтни типови преносе се по вриједности
Рад са	Програмер мора водити рачуна о	Виртуелна машина води рачуна о

меморијом	алокацији и делокацији меморије	меморији, без учешћа програмера
Вишеструко наслеђивање	Да	Не, могуће је наслиједити једну класу и више интерфејса

Табела 2. Основна својства програмских језика C++ и Јава

У првој врсти табеле 2, гдје смо говорили о преносивости, навели смо двије реченице под знацима навода, које вриједи додатно појаснити:

- Пиши једном, компајлирај свуда (енг. *Write Once, Compile Anywhere; WOCA*) односи се на писање софтвера који се може компајлирати, па према томе употребљив је, на свим платформама (оперативним системима). Међусобне разлике у оперативним системима, попут функција за приступ мрежи или графичком интерфејсу, превазилазе се употребом платформски независних корисничких библиотека. Преносивост се огледа на нивоу изворног кода, а не на нивоу преведеног програма, нпр. машинског кода.
- Пиши једном, покрећи свуда (енг. *Write Once, Run Anywhere; WORA*) је слоган који је осмислила компанија „Сан микросистемс“ (енг. *Sun Microsystems*) који се односи на особину програма писаних у јави да њихов бајткод (настао превођењем изворног кода у међујезик) може да се покрене на било ком рачунару, на било којој платформи. Ову особину Јава дугује виртуелној машини, засебној компоненти која мора бити инсталирана на циљном рачунару и која ће заправо опонашати понашање оперативног система, пружајући услуге процесора, меморије и других компонената рачунара у заштићеном режиму.

Иако Јава има предност у односу на C++ по питању преносивости, њена ефикасност се често доводи у питање у односу на C++. На интернету постоји велики број резултата различитих анализа по овом питању и тешко је наћи досљедност међу њима. У зависности од врсте теста који се спроведе, врсте апликације која се тестира и особе или организације која је спровела тест, могу се видјети сасвим опречни резултати. На примјер, у документу Улриха Стема са универзитета Стенфорд из 2001. године, детаљно тестирање брзине различитих алгоритама на разноврсним оперативним системима јасно показује значајну предност C++-а у односу на Јаву[3]. С друге стране, у документу „Јава у теорији и пракси: урбане легенде о перформансама, осврт“ (енг. *Java theory and practice: Urban performance legends, revisited*) говори се о супериорности механизма алокације виртуелне машине и опасностима које C++ носи собом[4]. Попут ових, коришћењем произвољног претраживача може се наћи низ аматерских тестова који такође показују опречне резултате.

Заговорници идеје да су програми писани у Јави ефикаснији од истих таквих писаних у C++-у образлажу ту идеју између осталог тиме да је виртуелна машина адаптивне природе, те је у стању детаљно анализирати оперативни систем и физичке особине рачунара прије него што покрене одређени програм. Програм писан у C++-у, међутим, углавном је компајлиран изван машине на којој ће радити, те оптимизације овог типа нису могуће. Поред тога, виртуелна машина има на располагању сву меморију која јој је дата од оперативног система, те је у стању вршити бољу компресију исте, што се све може подвести под аргументе дате у IBM-овом чланку изнад. На крају, позива се на извјесне језичке конструкције Јаве које боље помажу процесу

3 Детаљи тестирања доступни су на веб адреси verify.stanford.edu/uli/java_cpp.html

4 Цјелокупан чланак доступан је на www.ibm.com/developerworks/java/library/j-jtp09275/index.html

оптимизације, за разлику од C++-а који, због постојања показивача и директног приступа меморији, гдје су такве оптимизације сувише опасне.

Иако уважавам неке од датих аргумената, неке од особина Јаве непоколебљиво говоре у корист C++-а:

1. Само постојање виртуелне машине као слоја између оперативног система и апликације заузима извјесну количину меморије и процесорског времена, за разлику од апликације писане у C++-у, која се извршава у директној комуникацији са оперативним системом
2. Пишући програм у C++-у, програмер мора водити рачуна о алокацији и делокацији меморије. Ово је отежавајућа околност која, међутим, узрокује максимално искоришћење меморије, ако се обави добро (у супротном добијамо проблем познат као „цурење меморије“). Виртуелна машина, с друге стране, води рачуна о броју употреба одређеног меморијског сегмента (енг. *reference count*), као и уз помоћ других алгоритама предвиђања кад одређени меморијски сегмент више није потребан, што води ка одложеном ослобађању меморије, беспотребном захтијевању вишка меморије од оперативног система у критичном тренутку, као и потрошњи процесорског времена потребног за вршење оваквих процјена. Јава програмер нема могућност да води рачуна о алокацији и делокацији меморије, те о овом аспекту програмирања обично и не размишља, што води ка лошем искоришћењу меморије и у другим аспектима – бирању подесног алгоритма и структуре за одређени задатак.

На основу наведеног, закључујем да C++ има боља својства по питању ефикасности, а да се преносивост може довољно подржати одабиром подесне библиотеке за развој.

Библиотека за развој - Qt

Једна од познатих мултиплатформских библиотека за развој, уз то слободна (лиценца ЛППГЛ[5]), доступна за C++ и таква да подржава развој графичког корисничког интерфејса, јесте Qt (чита се „кјут“, „кјути“ или једноставно „ку-те“). Ову библиотеку првобитно је развила норвешка компанија „Тролтек“ (енг. *Trolltech*), а власништво над њом и даљи развој касније је преузела компанија „Нокија“, куповином компаније „Тролтек“ 2008. године. Читав низ високо-комерцијалних програма развијен је користећи ову библиотеку, попут: „*Autodesk Maya*“, „*Adobe Photoshop Elements*“, „*Skype*“, „*VLC media player*“ и „*Mathematica*“. Програм писан користећи овај софтверски пакет може се компајлирати и извршити на различитим оперативним системима; постоји подршка за виндоус, линукс, мек-ос, као и за различите мобилне платформе[6].

Основни језик ове библиотеке је C++, али постоји подршка и за друге програмске језике, првенствено пајтона, али и PHP-а, јаве и других.

Qt је више од обичне библиотеке за C++. Он представља проширење стандардног C++-а, а језичке конструкције које представљају то проширење могуће су захваљујући постојању припремног алата, под називом *qmake*. *Qmake* се обично покреће два пута прије него што се код преда одговарајућем стандардном компајлеру:

---

5 Комплетни подаци о лиценци за Qt могу се пронаћи на „Нокијином“ веб сајту, на веб адреси http://qt.nokia.com/products/licensing

6 Списак подржаних платформи видјети на адреси doc.qt.nokia.com/4.7-snapshot/supported-platforms.html

1. У првом кораку команда се покреће са само једном опцијом командне линије : –project, док смо позиционирани у кореном директоријуму нашег пројекта обогаћеног датотекама с изворним кодом. Овако позвана команда генерисаће пројектну датотеку са екстензијом .pro, на основу садржаја директоријума у ком је позвана. Пројектна датотека садржи параметре који су препознати на основу анализе пронађеног изворног кода, што подразумијева списак датотека са изворним кодом, као и подразумијеване заставице које ће бити коришћене доцније, у процесу компајлирања. Ова датотека може се уређивати произвољним едитором текста прије него што се пређе у наредну фазу.

2. У другој фази поново се позива команда qmake, али овај пут јој се просљеђује име малочас генерисане пројектне датотеке као први параметар командне линије. Сада се генеришу датотеке са изворним кодом на основу .ui датотека које представљају дефиницију графичког корисничког интефејса, датотеке са изворним кодом које пружају потпору горе-поменутим језичким проширењима (тзв. .moc датотеке) и Makefile за униксолике системе (укратко: датотека која дефинише редослијед и зависности у процесу компајлирања; изван досега овог рада), односно пројектна датотека за Мајкрософтов „Visual Studio“. Након овог стадијума, пројекат је спреман за компајлирање користећи одговарајући компајлер за стандардни C++.

Програмирање у библиотеци Qt је *вођено догађајима*. То значи да програмер користи језичка проширења тако да објекти могу *емитовати догађаје* (сигнале) и функције које ће *реаговати на те догађаје* (слотове). То се чини користећи посебну функцију connect из класе QObject, која је основна у овој библиотеци и скоро све уграђене класе изведене су из ње. На примјер, да би се направило дугме које врши одређену функцију, дефинише се њен сигнал „кликнуто“ и повезује с корисничком функцијом која представља руковаоца овог догађаја:

```
connect(button, SIGNAL(clicked()), handler, SLOT(handleClicked());
```

Механизам помоћу којег је подржано руковање догађајима назива се „петља догађаја“ (енг. *event loop*), слична Мајкрософтовој „петљи порука“ (енг. *message loop),* која се одвија у оквиру главне нити програма и обрађује пристижуће догађаје. Важно је при томе обратити пажњу да ће се и обрада конкретних догађаја дешавати у главној петљи, те треба водити рачуна да та обрада траје довољно кратко да и други догађаји могу бити благовремено обрађени, у првом реду исцртавање графичког корисничког интерфејса. Уколико је потребна сложенија и захтјевнија обрада која не може трајати тако кратко, препоручује се да се та обрада обави у засебној нити, да не би дошло до „замрзавања“ прозора апликације.

Да би се добила петља догађаја у главној нити, потребно је креирати објекат класе QApplication (или QCoreApplication за конзолне програме) било гдје у програму, те позвати његову методу exec. Ово се обично чини већ у главној функцији:

```
#include <QApplication>
int main()
{
  QApplication app;
  return app.exec();
}
```

Обично се у главној функцији формира и приказује објекат који представља главни прозор апликације, а одмах потом се улази у петљу догађаја апликације помоћу функције exec на објекту класе QApplication, да би тај прозор могао добијати догађаје од

оперативног система (улазни/излазни уређаји). Ако главни прозор не очекује одређене догађаје, о њима ће се побринути подразумијевана имплементација апликације.

```
#include <QApplication>
#include "MyMainWindow.h"
int main()
{
  QApplication app;
  MyMainWindow wnd;          // формирамо објекат за главни прозор апликације
  wnd.show();                                                  // приказујемо прозор
  return app.exec();                                  // улазимо у петљу догађаја
}
```

Графички кориснички интерфејс (који обухвата, на примјер, главни прозор апликације, споменут изнад) подржан је у основи Qt-а и постоји низ уграђених графичких алата који помажу развоју истог:

- Qt Designer; програм који омогућава „цртање“ графичких контрола и прозора
- Qt Linguist; програм који омогућава локализацију графичког интерфејса
- Qt Creator; интегрише претходне алатке, уз текстуални едитор и интегрисано компајлирање и организовање пројекта

Qt Creator сличне је намјене као конкурентски производ Мајкрософта, „MS Visual Studio“. У њему се може направити пројекат који води рачуна о свим припадајућим датотекама, приступити интегрисаном систему помоћи који садржи детаљан опис свих доступних класа и функција, цртати кориснички интерфејс, компајлирати пројекат, дебаговати и покренути његова извршна верзија. Говоримо, дакле, о комплетном развојном окружењу.

Qt Linguist може се употријебити на оним пројектима који су припремљени за локализацију. Припрема за локализацију у оквиру изворног кода одвија се крајње једноставно:

1. Ако у самом изворном коду дефинишемо ниске карактера које ће бити приказане кориснику, дакле захтијевају локализацију, потребно их је предати функцији QObject::tr (скр. енг. *translate*), као у сљедећем примјеру: tr("text to localise"). Све текстуалне поруке дефинисане у графичком корисничком интерфејсу (дакле, користећи Qt Creator) аутоматски су припремљене за превођење. Уобичајено се поруке у самом изворном коду пишу на енглеском језику, који данас представља *lingua franca*, а за остале језике се обезбјеђује превод.

2. Након што је програм завршен, може се направити датотека за превод ниски дефинисаних у њему. Ова датотека носи екстензију .ts (скр. енг. *translation)* и формира се тако што се покреће команда lupdate, којој се као први аргумент командне линије предаје име пројектне датотеке. У њој се јављају све преводиве ниске карактера дефинисане у кораку 1, а за њихов превод на одговарајући језик дефинисане су празне ниске које се попуњавају уз помоћ програма Qt Linguist.

3. Кад имамо спреман превод на одговарајући језик, он се „објављује“ уз помоћ команде lrelease, чиме се формира бинарна датотека са екстензијом .qm, спремна за употребу. Уколико име ове датотеке представимо као превод програма, програм ће је аутоматски учитати у на одговарајућим мјестима приказаће се преведени текст.

За наше потребе најзанимљивији је **графички модул** Qt-а под званичним називом „Graphics View Framework“. Овај модул омогућава да се у кориснички интерфејс постави графичка контрола способна за приказ основних равних геометријских фигура,

динамички креираних уз помоћ ограниченог броја класа и припадајућих метода. Qt подржава и рад са библиотеком OpenGL, али пошто она првенствено служи развоју апликација за руковање тродимензионалним објектима и беспотребно би допринијела нивоу сложености апликације, користићемо прву поменуту опцију. У поглављу „Архитектура“ ниже, биће више ријечи о детаљима овог модула за графику.

Qt садржи и модул за **рад са улазним и излазним аудио уређајима**. Апстракција улазног и излазног аудио уређаја садржана је у оквиру класа QAudioInput и QAudioOutput, уз помоћ којих на једноставан начин можемо затражити одговарајуће услуге од оперативног система. Ове класе подржавају подешавање свих параметара који су неопходни за снимање и репродукцију звука: фреквенција, број канала, кодек, поредак бајтова (енг. *endianness*) итд. Qt такође подржава испитивање циљаног оперативног система и на њему подржаних аудио уређаја, тако да поменути параметри нису ~~„амин“~~дати једном заувек: у току свог рада програм може замолити оперативни систем да му да „најближе“ параметре, ако параметри које је он дефинисао нису у потпуности подржани. На тај начин правовремено можемо обавијестити корисника да систем није у потпуности подржан, те да репродукција или снимање неће бити оптимални.

Архитектура пројекта

Да бисмо дефинисали архитектуру пројекта, потребно је да проучимо особине развојног окружења које смо изабрали. Нама најбитнија компонента одабраног развојног окружења, као што смо истакли у претходном поглављу, јесте модул за рад са графичким објектима, Graphics View. Говорићемо о томе како је он конципиран.

Графичка контрола која се може интегрисати у главни прозор програма омогућена је класом QGraphicsView. Обратити пажњу да се ова класа зове као и цјелокупан модул, тј. да су у питању двије различите компоненте. QGraphicsView може бити произвољне величине, положаја и декорације и служи само као „излаз“ позадинске логике на екран.

Апстракција „сцене“, као виртуелног простора на ком леже графички објекти, садржана је у класи QGraphicsScene. У контексту хијерархије графичких објеката, у којима један објекат може бити „родитељ“ другом (садржи га или је његов творац), сцена служи као родитељ свим објектима који немају другог родитеља. Она садржи методе за додавање објекта (addItem), уклањање (removeItem), претрагу (items) и трансформацију из једног координатног система у други (мапирајуће функције). Координатни систем сцене назива се *глобални координатни систем*.

Сваки графички објекат на сцени мора припадати класи насљедници класе QGraphicsItem. Њу насљеђује класа QGraphicsObject, која додатно има својства емитовања и обраде догађаја, описана изнад. Координатни систем једног објекта на сцени назива се његовим *локалним координатним системом.* И једна и друга класа имају сљедеће апстрактне функције, које је потребно имплементирати у свим изведеним класама, јер од њих зависи изглед и функционалност дотичног објекта:

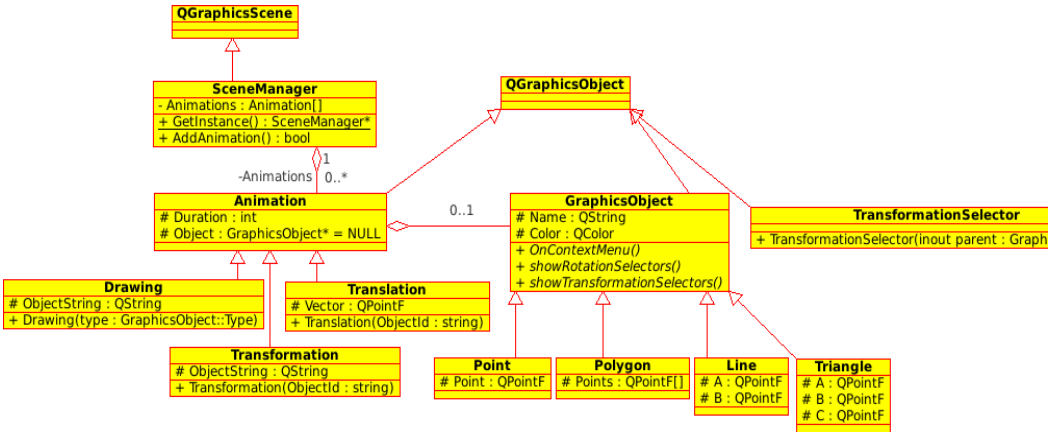
- QRectF boundingRect() const; генерише правоугаону површину у оквиру сцене која ће бити заузета дотичним објектом. Правоугаоник треба да узима у обзир глобалне координате сцене.
- void paint(QPainter *painter, ...); врши само цртање објекта, на објекту painter, који представља „платно“ сцене. Цртање се одвија у локалном координатном систему. Класа QPainter садржи методе за цртање елементарних геометријских фигура (елипсе, многоугла, правоугаоника, дужи, тачке и текста), као и методе за контролу боје њихових ивица и позадине.

– – – – – – – – –

У складу с изложеним чињеницама и као производ вишедневног разматрања, аутор овог рада сматра да кључне компоненте будућег система треба да буду сљедеће:

- **Менаџер сцене**, класа изведена из класе QGraphicsScene; руковаће сценом коју обухвата и свим припадајућим објектима. Он ће вршити валидацију додатих објеката, претрагу, образовање XML документа на основу текућег стања сцене и репродуковати стање сцене на основу датог XML документа. Уз његову помоћ вршиће се анимирање објеката на сцени.
- **Класе за графичке објекте**, које обухватају: угао, круг, елипсу, дуж, тачку, много-угао, правоугаоник, текст и троугао. Биће изведене из базне класе „графички објекат“, која ће бити изведена из класе QGraphicsObject. Међукласа је додата зато што ће сви наши објекти имати одређена својства и функционалности који нису обухваћени подразумијеваном имплементацијом Qt-овог графичког објекта.
- **Ручице за трансформацију објеката** (при процесу ротације, трансформације и транслације) такође ће бити графички објекти. Ручице ће бити покретне и графички објекти на које се оне односе пратиће њихово кретање и реаговати одговарајућом трансформацијом.
- **Анимације** ће такође бити имплементиране као графички објекти. Наиме, свака анимација ће имати, поред саме референце на објекат који се мијења, и друге пратеће декорације, почев од споменутих ручица. Анимација за цртање биће власник дотичног графичког објекта, а друге анимације имаће само референцу на њега у циљу упућивања инструкција за трансформацију.
- **Главни прозор** биће централни ентитет апликације, који ће спајати све наведене и ненаведене елементе дизајна: у његовом средишту биће графички преглед у чијој позадини ради менаџер сцене; клик мишем на алатке за цртање узроковаће да се на сцену дода анимација цртања оног објекта на које се односи дотично дугме; историја анимација ће мотрити стање сцене и одражавати га графички; главни прозор ће водити рачуна о звуцима убаченим у текући документ и захтијевати од менаџера сцене да обави интеракцију са садржајем датотеке коју корисник изабере у циљу снимања, отварања или репродукције.
- Постојаће два додатна прозора са сопственом логиком: прозор за снимање звука, који ће користити модул за рад са улазним и излазним аудио уређајима, описан раније у поглављу „Библиотека за развој – Qt“, те прозор за подешавање корисничких опција, који ће снимати задата подешавања у иницијализациону датотеку, регистар или трећи ентитет зависан од конвенције оперативног система.

На слици 8. може се видјети UML дијаграм основних класа које ће бити коришћене у програму.



Слика 8. Скраћени UML дијаграм основних компоненти система

Имплементација

Да би се развио одређени пројекат, битно је у самом почетку увести систем контроле ревизија. Као што нам већина текстуалних едитора нуди могућност поништавања посљедњег корака (често унатраг до првог извршеног корака и обратно), тако нам систем контроле ревизија нуди могућност праћења свих измјена на пројекту, почев од прве линије кода коју смо написали до завршетка пројекта, независно од било каквог текстуалног едитора који смо користили. Још битније, али мање битно за наше потребе, систем контроле ревизија нуди могућност колаборације више учесника пројекта.

Системи контроле ревизија

Да бисмо објаснили шта је то систем контроле ревизија и како функционише, размотримо један примјер.

Примјер 1. Милан и Драган развијају софтверски пројекат живећи у различитим градовима. Имају сервер на интернету на ком се налази комплетан изворни код и ком обојица имају приступ. О битним одлукама договарају се путем аудио и видео конференције. Дешавају им се сљедећи проблеми:

- Драган се пријављује на сервер и отвара датотеку A.cpp. Након извршених измјена, покушава да сними датотеку и открива да се њен садржај у међувремену промијенио, јер је Милан отворио исту датотеку и унио измјене прије него што је Драган завршио. Приморан је да отпише своје измјене, а потом их поново нанесе на поновно отворену датотеку с Милановим измјенама. По свршетку поновног наношења измјена проблем се може појавити у истом облику.
- Милан прегледа листинг директоријума A и на основу података о времену посљедње измјене увиђа да је Драган вршио неке модификације на датотеци A.cpp; хтио би да види које су то модификације, али има на располагању само комплетан садржај датотеке. Он позива Драгана неким од комуникационих канала, али Драган а) није доступан б) може да се сјети само опште идеје својих измјена.
- Милан је био јако вриједан и током ноћи извршио огромну количину измјена на

различитим датотекама на серверу. Сутрадан схвата да је направио крупну грешку у концепцији, да измјене нису валидне, те да је потребно вратити садржај датотека на старо стање. Покушава да врати податке из бекапа, али бекап је формиран прије више дана и пројекат се враћа неколико дана унатраг.

Да би се ријешили овакви и слични проблеми на које се наилази током рада на колаборативном пројекту, осмишљен је концепт ревизије и репозиторијума.

Ревизија представља скуп измјена на једној или више датотека, које је направио један аутор у циљу постизања одређеног задатка. Аутор све вријеме ради на *копијама* датотека, а по завршеном послу уз помоћ специјализованог софтвера шаље на репозиторијум само *измјене* које је направио на копијама.

Репозиторијум је у својој суштини сам пројекат на ком се ради и као такав представља централно мјесто колаборације. Учесници пројекта овдје шаљу своје ревизије. Само по себи је јасно да све ревизије заједно формирају посљедње стање пројекта у цјелини.

Важно је нагласити да учесници пројекта не шаљу комплетне датотеке које су измијенили, него само *измјене* на тим датотекама, односно разлике између старог и новог стања. Те разлике се изражавају на начин који је својствен одговарајућем систему контроле ревизија, који постоје у различитим форматима и комерцијалним или слободним пакетима. Неки од најпознатијих су: CVS (concurrent versions system), SVN (subversion), git (the stupid content tracker).

Помоћу система контроле ревизија између осталог могуће је:

- Добити преглед учињених измјена на једној или више датотека, скупа с подацима о аутору измјена, датуму и времену, коментару који је унио скупа с измјеном и сл.
- Поништити појединачну измјену или скуп измјена, да би се дотичне датотеке вратиле на неко претходно стање.
- Регулисати ситуацију када се „сударе“ измјене два или више корисника (конфликти). То се дешава када два или више корисника преузму копију исте датотеке и направе измјене које су у сукобу једна с другом. Систем може сам регулисати ситуације када учесници на истој датотеци праве измјене које нису у сукобу, али када јесу, учеснику који наиђе на проблем нуди се да изабере коју од измјена да прихвати или да их ручно споји у логичну цјелину.
- Добити преглед ауторства сваког појединачног реда текста у одређеној датотеци (енг. blame). Код ове опције је проблематично што се заправо мјери посљедња измјена, те је довољно да неко измијени једно слово у датом реду, да би систем тај ред препознао као његово ауторство.

За рад једног аутора на пројекту, на личном рачунару, нарочито је популаран слободни и једноставни алат за контролу ревизија, git. Он нуди већину функционалности које нуде и остали системи за контролу ревизија, а његова употреба крајње је једноставна. Постоји верзија за рад из командне линије, као и графичко окружење.

Формирање пројекта за цртање – дизајнер

Први задатак у формирању пројекта (дизајнера и прегледача подједнако) биће формирање репозиторијума. Приказаћемо команде помоћу којих смо направили репозиторијум за дизајнер:

```
$ mkdir Master
```

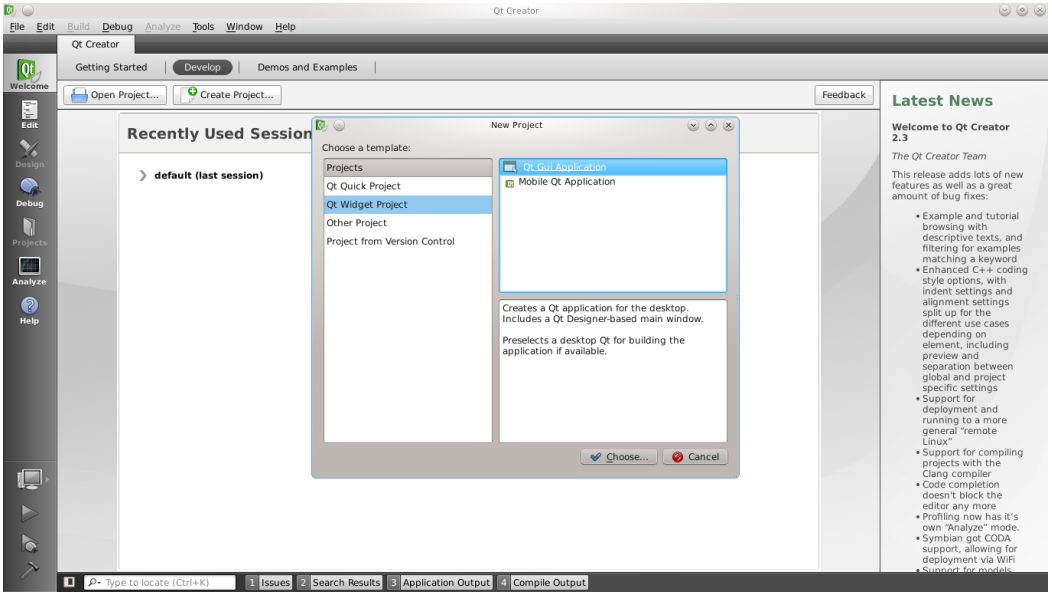
```
$ cd Master
$ mkdir Designer
$ cd Designer
$ git init
Initialized empty Git repository in /home/darko/Master/Designer/.git/
```

Обратимо пажњу да скривени директоријум .git представља мјесто гдје се чувају све измјене (репозиторијум), док ће све датотеке које овдје будемо писали и додавали у репозиторијум представљати само копије на којима се ради. Ово је занимљиво јер већина других система за контролу ревизија претпоставља да ће репозиторијум бити на удаљеном серверу којем ће сви учесници пројекта приступати преко мреже, чувајући на локалном рачунару само копије које добаве са сервера.

Сваку фазу пројекта или завршен засебан задатак биће потребно отпремити у репозиторијум, како би будуће измјене припадале новим ревизијама које можемо разликовати и упоређивати. То се постиже сљедећом наредбом:

```
$ git commit
```

Да бисмо отпочели пројекат писан помоћу библиотеке Qt, најбоље је да користимо њену компоненту Qt Creator (слика 9).



Слика 9. Qt Creator; приказан је дијалог за стварање новог пројекта. На слици се види да смо одабрали апликацију са графичким корисничким интерфејсом.

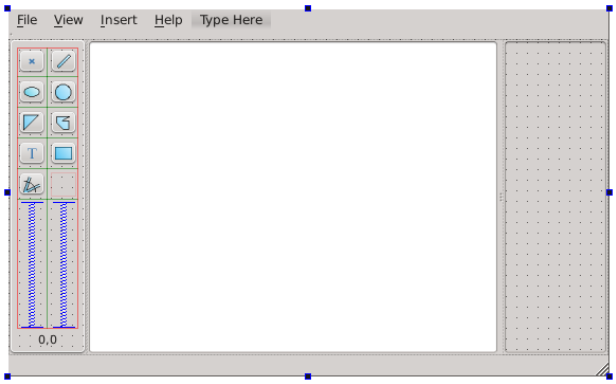
Овај алат нам омогућава да одаберемо тип пројекта (у нашем случају, апликација са графичким корисничким интерфејсом – енг. GUI Application), а он ће нам направити његов костур са основном функционалношћу. Костур обухвата сљедеће:

1. main.cpp; овдје се чува главна функција, од које почиње извршење програма.
2. MainWindow.ui; празан графички прозор у који можемо додати нове компоненте.
3. MainWindow.cpp и MainWindow.h; класа MainWindow, која ће пружити функционалност графичком прозору из ставке 2.
4. ImeProjekta.pro; пројектна датотека, формирана аутоматским позивом команде

qmake као што је описано у кораку 1. поглавља „Библиотека за развој – Qt“.

Qt Creator такође има уграђену подршку за git – по свршетку креирања новог пројекта, систем ће нас питати да ли желимо да додамо датотеке у репозиторијум. Пошто одговоримо са „да“, све нове датотеке побројане у тачкама изнад биће додате у систем контроле ревизија. Даље ће бити неопходно само користити ставке из једног од менија Qt Creator-a да би се руковало репозиторијумом.

Први корак у креирању програма са графичким корисничким интерфејсом требало би да буде креирање самог корисничког интерфејса. Овакав почетни положај даће нам могућност да тестирамо прве функционалности које имплементирамо и правовремено имамо евиденцију како различите функционалности интерагују једна с другом. На слици 10. може се видјети припремљен интерфејс у складу са концепцијом из поглавља „Радни простор“.



Слика 10. Графички кориснички интерфејс на којем ћемо базирати пројекат.

На слици 10. може се примијетити да смо писали текст на енглеском језику, а преводи на друге језике (укључујући у првом реду српски) биће додати накнадно, локализацијом. Од овог тренутка, па до финализације пројекта, биће писан само одговарајући изворни код који ће „оживјети“ графички кориснички интерфејс изнад. Приступиће се писању конкретних класа које смо дефинисали у поглављу „Архитектура“, почев од сценског менаџера, преко класа за анимације и графичке објекте, до класа које их спајају и помажу им у раду.

Укупна величина изворног кôда пројекта је 588 килобајта, од чега око 350 килобајта за дизајнер и око 240 килобајта за прегледач, при чему је релативно велики дио кода заједнички за оба програма. Мјерено у страницама А4 формата, ово износи око 355 страница текста. Из тог разлога овај рад не може бити мјесто гдје ће бити приказан цјелокупан изворни кôд. Биће приказани само кључни дијелови, у циљу кратког прегледа појединачних корака развоја. Из тог разлога главнину приказаног кôда чиниће дефиниције класа, а имплементација појединачних функција биће приказана само у оној мјери која је неопходна. Из истих разлога биће изузете функције које су уведене само у циљу скраћења кôда, извођења тривијалних операција итд. Изворни кôд у цјелини доступан је у прилогу.

Изворни кôд апликације

Главни прозор

Приказ изворног кода отпочећемо од класе која даје функционалност главном прозору – MainWindow. Наиме, у овој класи дефинисане су методе које ће бити прве позване када корисник пожели да обави било коју операцију. Због тога ће читаоцу овог рада бити природно да одатле прати логику инструкција које су неопходне за извршење тих операција.

Дефиниција класе MainWindow може се видјети у прилогу 2.

```
class MainWindow : public QMainWindow
{
public:
   MainWindow(QWidget *parent = 0);
   ~MainWindow();
   QVector<SoundStruct> getSoundsList();
private:
   Ui::MainWindow *ui;
   PaletteManager *paletteManager;
   GraphicsObject::Type drawnObjectType;
   void newObject(GraphicsObject::Type type, QToolButton *button);
   QString m_filename;
   QVector<SoundStruct> m_sounds;
   QMutex m_soundLoaderMutex;
public slots:
   void save();
   void saveAs();
   void open();
   void newDoc();
   void showAbout();
   void showOptions();
   void showRecording();
   void previewAnimation();
   void statusText(const QString &text);
protected slots:
   void newPoint();
   void newLine();
   ...
   void newAngle();
   void animationComplete(Animation*);
   void animationCanceled(Animation*);
   void errorLoadingSoundsDuringOpen(QString);
};
```

Прилог 2. Изворни кôд класе која рукује главним прозором

Методе класе MainWindow могу се подијелити у двије главне групе: методе које

служе за додавање новог објекта на сцену (newText, newCircle итд.) и методе које покрећу друге компоненте графичког корисничког интерфејса, попут дијалога за снимање и отварање датотеке, дијалога за снимање звука и дијалога за подешавања. Поред њих, ту су још двије методе – animationComplete и animationCanceled – које реагују на сигнале које пошаље графичка сцена у циљу ажурирања графичких контрола.

Методе које служе за додавање новог објекта на сцену, пошто слиједе исту логику са различитим типовима објеката, све позивају методу newObject. Ова метода креира анимацију цртања за одговарајући тип графичког објекта (нпр. за цртање елипсе биће алоциран нови AnimationDrawing(GraphicsObject::EllipseType)), а затим га предаје менаџеру сцене. Описани поступак, тј. имплементација методе MainWindow::newObject, може се видјети у прилогу 3.

```
void MainWindow::newObject(GraphicsObject::Type type, QToolButton *button)
{
  try
  {
    if (drawnObjectType == type)
    {
      button->setChecked(true);
      return;
    }
    Animation *newAnimation = new AnimationDrawing(type);
    if (SceneManager::getInstance()->addAnimation(newAnimation))
    {
      button->setChecked(true);
      drawnObjectType = type;
    }
    else
    {
      button->setChecked(false);
      delete newAnimation;
    }
  }
  catch (const QString &msg)
  {
    QMessageBox::critical(this, tr("Error"), msg);
    QApplication::exit(1);
  }
}
```

Прилог 3. Метода newObject класе главног прозора

Начин убацивања и контроле новог објекта у оквиру менаџера сцене биће описан ниже, у поглављу „Менаџер сцене“. Прије тога погледајмо кратак опис метода главног прозора које служе за покретање других компонената графичког корисничког интерфејса:

- save, saveAs, open, newDoc; све служе за покретање уграђеног дијалога за одабир датотеке у оквиру датотечног система. Библиотека Qt садржи класу QFileDialog,

која дохвата подразумијевани системски дијалог за одабир датотеке, те ће корисник наићи на себи познат интерфејс.
- showAbout, showOptions, showRecording; приказују дијалоге који су креирани у оквиру овог пројекта, редом: дијалог за приказ основних података о програму, дијалог за подешавања програма и дијалог за снимање звука. Логика појединачних дијалога имплементирана је у оквиру одговарајућих класа, а одавде се само шаље инструкција да се они прикажу.
- previewAnimation; снима тренутну сцену у привремену датотеку и покреће спољашњу апликацију – прегледач – да репродукује досадашње анимације.
- statusText; приказује помоћни текст кориснику у статусној траци. Ова трака се користи за инструкције кориснику у току цртања (нпр. „кликните још једном да бисте дефинисали центар елипсе“).

Менаџер сцене

Менаџер сцене заправо представља саму сцену и њеног контролера у истом. Будући да има тако високу функцију, други објекти (на првом мјесту анимације и графички објекти, али и главни прозор) често имају потребу да затраже неку од његових функционалности. С обзиром да је у оквиру апликације присутна само једна графичка сцена, ово је могуће ријешити користећи дизајн „синглтон“ (енг. singleton). То значи сљедеће:

- Може постојати само једна инстанца класе у оквиру једног покретања програма
- Класа садржи јавну статичку методу getInstance, која креира (ако већ није креирала) и враћа једину инстанцу те класе
- Једина инстанца класе дефинисана је као статички члан класе
- Конструктор класе је декларисан у приватној секцији

На овај начин довољно је да је у одређеном дијелу кода позната дефиниција класе SceneManager и моћи ће се приступити тој њеној јединој инстанци која представља графичку сцену на екрану. Менаџер сцене нуди сљедеће функционалности:

- Додавање нових анимација (цртање новог објекта или трансформација над постојећим објектом) и уклањање постојећих. Методе: addAnimation, removeAnimation, pop и clear.
- Руковање догађајима са улазних уређаја (конкретно: миш и тастатура) и њихова дистрибуција тренутно активним објектима (анимацијама и графичким објектима на сцени). Методе: mouseMoveEvent, mousePressEvent, mouseDoubleClickEvent, keyPressEvent и keyReleaseEvent. Међу овим методама је и contextMenuEvent, која реагује када корисник захтијева контекстуални мени. Издвојена је засебно од методе за руковање догађајима миша, јер може бити изазвана и коришћењем специјализованог тастера на тастатури.
- Креирање XML документа на основу тренутног стања сцене и репродукција старог стања сцене на основу XML документа – методе toXML и recoverFromXML. Ове двије методе у основи су функцијских позива openDoc, save и saveAs у класи главног прозора. Оне даље позивају истоимене методе класа изведених из Animation, а оне позивају такве исте методе класа изведених из GraphicsObject
- Метода која проналази објекат „близу“ датог објекта, а која се користи при цртању.

Дефиниција класе SceneManager може се видјети у прилогу 4.

```
class SceneManager: public QGraphicsScene
```

```
{
public:
   static SceneManager *getInstance();
   virtual ~SceneManager();
   bool pop(Animation *);
   bool addAnimation(Animation *animation);
   QPointF findSnapped(const QPointF &point, Animation *requester);
   void toXML(QDomElement &animations, QDomDocument &doc) const;
   bool recoverFromXML(const QDomElement &animations, QStringList &errors);
   void clear();
   enum RemoveResult { Success, Dependencies, NoSuchAnim };
   RemoveResult removeAnimation(Animation *animation, bool force = false, int *inDepCount);

protected:
   QVector<Animation*> m_animations;
   virtual void keyPressEvent(QKeyEvent *keyEvent);
   virtual void keyReleaseEvent(QKeyEvent *keyEvent);
   virtual void mouseMoveEvent(QGraphicsSceneMouseEvent *mouseEvent);
   virtual void mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent);
   virtual void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *mouseEvent);
   virtual void contextMenuEvent(QGraphicsSceneContextMenuEvent *event);

private:
   SceneManager(QObject *parent = NULL);
   static SceneManager *instance;
   Animation *current;

public slots:
   void animationComplete();
   void animationCanceled();

signals:
   void animationCreated(Animation *currentAnimation);
   void animationCanceled(Animation *currentAnimation);
   void animationComplete(Animation *newAnimation);
   void mouseMoved(MousePosition);
};
```

Прилог 4. Дефиниција класе SceneManager

Најважнија функција у овој класи је свакако addAnimation. Она ће одобрити додавање нове анимације ако нека анимација није већ у процесу стварања, припремити сцену за цртање новог објекта и припремити анимацију за интеракцију са корисником. Такође ће обавијестити све заинтересоване стране да је креирана нова анимација. Изворни код ове методе може се видјети у прилогу 5.

```
bool SceneManager::addAnimation(Animation *newAnimation)
{
```

```
  if (current == NULL)
  {
    current = newAnimation;
    m_animations.push_back(current);                    // похрањујемо је у своју колекцију
    addItem(current);                                   // метода из QGraphicsScene
    connect(current, SIGNAL(complete()), this, SLOT(animationComplete()));
    connect(current, SIGNAL(canceled()), this, SLOT(animationCanceled()));
    current->initializeForInteraction();            // припрема за интеракцију с корисником
    emit animationCreated(current);             // обавјештење заинтересованим странама
    return true;
  }
  return false;
}
```

Прилог 5. Метода за додавање анимације у оквиру менаџера сцене

Метода initializeForInteraction уведена је јер анимације не морају одмах по свом стварању постати интерактивне; примјер је репродукција анимација на основу XML документа. У случају анимације за цртање новог објекта, ова метода није од нарочитог значаја, али у случају других типова анимације, њена првенствена улога је да се формирају ручице за трансформацију које корисник може помјерати.

Методе које рукују догађајима са улазних уређаја углавном само позивају подразумијевану имплементацију библиотеке. Међутим, у случају да је нека анимација тренутно у процесу формирања, она преузима обраду ових догађаја. Примјер може бити метода за руковање догађајем клика на дугме миша, у прилогу 6.

```
void SceneManager::mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent)
{
  if (mouseEvent->button() != Qt::LeftButton)
    QGraphicsScene::mousePressEvent(mouseEvent);            // подразумијевана импл.
  else if (current == NULL)
    QGraphicsScene::mousePressEvent(mouseEvent);            // подразумијевана импл.
  else if (!current->mousePressEvent(translateMouseEvent(mouseEvent), keyModifiers))
    QGraphicsScene::mousePressEvent(mouseEvent);            // подразумијевана импл.
  else
    mouseEvent->accept();
}
```

Прилог 6. Руковање догађајем клика на дугме миша у оквиру менаџера сцене

Метода за снимање тренутног садржаја сцене врло је једноставна. Она прихвата параметар који представља XML документ и покушава га попунити релевантним садржајем. Овај задатак она препушта објектима који представљају анимације, тако што у петљи замоли једног по једног да дају свој опис у XML форми.

Метода за репродукцију старог стања сцене на основу XML документа нешто је сложенија. Она мора сама испитати сваки XML чвор који представља анимацију, препознати његов тип, креирати анимацију тог типа, а затим тој анимацији препустити

даљи рад као у случају снимања. Додатно, ова метода сваку анимацију повезује са менаџером сцене преко догађаја complete и canceled, те и сама обавјештава заинтересоване стране да је додата нова анимација на сцену. Заинтересована страна биће на првом мјесту класа за главни прозор, која ће пожељети да ажурира прозор који приказује историју анимација. Метода recoverFromXML може се видјети у прилогу 7.

```
bool SceneManager::recoverFromXML(const QDomElement &animations, QStringList& errors)
{
  if (animations.isElement())
  {
    QDomElement animation = animations.firstChildElement("animation");
    while (!animation.isNull())
    {
      QString type = animation.attribute("type", "NO_VALUE");
      if (type != "NO_VALUE")
      {
        Animation *a = NULL;
        if (type.compare("drawing", Qt::CaseInsensitive) == 0)
          a = new AnimationDrawing;
        else if (type.compare("transformation", Qt::CaseInsensitive) == 0)
          a = new AnimationTransformation;
        else if (type.compare("rotation", Qt::CaseInsensitive) == 0)
          a = new AnimationRotation;
        else if (type.compare("translation", Qt::CaseInsensitive) == 0)
          a = new AnimationTranslation;

        if (a)
        {
          if (!a->recoverFromXML(animation, errors))
            return false;
          else
          {
            m_animations.push_back(a);
            addItem(a);
            connect(a, SIGNAL(complete()), this, SLOT(animationComplete()));
            connect(a, SIGNAL(canceled()), this, SLOT(animationCanceled()));
            emit animationComplete(a);
          }
        }
      }
      animation = animation.nextSiblingElement("animation");
    }
    update();
    return true;
  }
}
```

Прилог 7. Метода менаџера сцене која служи за репродукцију стања сцене на основу XML документа

Анимације

Централни појам у апликацији је објекат анимације. Он се користи за све основне акције у програму – цртање, ротирање, транслацију и трансформацију графичких објеката. Као што смо раније приказали у УМЛ дијаграму, објекат анимације имаће референцу на графички објекат на који се односи. Приказаћемо изворни кôд базне класе Animation у прилогу 8.

```
class Animation: public QGraphicsObject
{
public:
   Animation(int duration = 0, GraphicsObject *target = NULL);
   virtual ~Animation();
   virtual bool initializeForInteraction() {return false;}

   virtual bool mouseMoveEvent(...) = 0;
   virtual bool mousePressEvent(...) = 0;
   virtual bool mouseDoubleClickEvent(...) { return false; }
   virtual void keyPressEvent(...) {}
   virtual void keyReleaseEvent(...) {}
   virtual void contextMenuEvent() {}

   virtual QWidget* getPaletteFrame() = 0;
   virtual void deletePaletteFrame() = 0;
   virtual HistoryItem *getHistoryWidget();
   virtual QPainterPath shape() const {return QPainterPath();}
   virtual bool snap(const QPointF &point, QPointF *snapPoint, qreal *distance) const;

   virtual QDomElement toXML(QDomDocument& root) const = 0;
   virtual bool recoverFromXML(QDomElement& animation, QStringList& errors) = 0;

   virtual void setPercent(qreal percent) = 0;
   virtual qreal getPercent() const = 0;
   virtual void revert() = 0;
   virtual void reapply(qreal percent = 100) = 0;


signals:
   void complete();
   void canceled();
   void statusText(QString);


protected slots:
   void animationCanceled();


protected:
   int m_duration;
   int m_pause;
   GraphicsObject *m_target;
   QDialog *m_frame;
   qreal m_percent;
};
```

Прилог 8. Дефиниција класе Animation, базне за све врсте анимација

Искористићемо овај тренутак да обратимо пажњу на специфичне инструкције које подржава Qt, а који су били коришћени и у раније наведеним класама. Објекат

анимације биће у стању емитовати информацију да је корисник саопштио да је у потпуности дефинисао анимацију (сигнал complete) или да ју је отказао (canceled). У циљу приказивања савјета за цртање, анимација може емитовати и произвољан текст (statusText) који ће бити приман у главном прозору и приказиван у статусној траци. С друге стране, постоји и један слот – animationCanceled – који служи за реаговање на догађај када корисник притисне дугме „одустани“.

Поред наведених сигнала и слотова, у овој класи можемо примијетити четири групе метода:

- Методе наслијеђене из класе QGraphicsObject, које дефинишу понашање класе у случају интеракције корисника с апликацијом (методе које покривају акције мишем и тастатуром)
- Методе које генеришу графичке контроле које служе за прецизно дефинисање параметара објекта и приказ анимације као ајтема у оквиру историје анимација (getPaletteFrame и getHistoryWidget)
- Методе које се односе на складиштење документа, односно метода која на основу параметара објекта генерише XML који их описује и метода која на основу датог XML документа реконструише састав објекта
- Методе које служе за постављање анимације у одређени стадијум, односно методе које служе за поништавање ефекта анимације и методе за постављање анимације у одређени стадијум на основу датог процента

Метода која је задужена за успостављање режима интеракције између корисника и сцене јесте initializeForInteraction. Спомињали смо је раније, у поглављу „Менаџер сцене“, а овдје ћемо погледати један њен примјер. Изабраћемо класу AnimationRotation и приказати изворни код имплементације методе initializeForInteraction, у прилогу 9.

```
bool AnimationRotation::initializeForInteraction()
{
  m_target = GraphicsObject::getObjectByName(m_targetName);
  if (m_target != NULL)
  {
    if (m_mirror == NULL)
      m_mirror = m_target->clone();

    m_mirror->objectCanceled();                    // објекат-огледало је „незавршен“
    m_target->setPenStyle(Qt::DotLine);   // огледало истачкано, оригинал остаје исти
    m_mirror->setMirrorOf(m_target);              // ВАЖНО – огледало зна да је огледало
    m_handles = m_mirror->getRotationHandles(); // стварамо ручице трансформације
    RotationHandle *centerHandle = new RotationHandle(transformOriginPoint(), -1);
    m_handles.prepend(centerHandle);
    for (int i = 0; i < m_handles.size(); i++)
    {
      connect                 // бићемо обавијештени сваки пут кад се помјери нека ручица
      (
m_handles[i], SIGNAL(moved(Handle*)),
this, SLOT(rotationHandleMoved(Handle*))
      );
      m_handles[i]->setOwner(findOwnerAnimation(m_targetName));
      if (scene() != NULL)
      {
```

```
            m_handles[i]->setPos(mapToScene(m_handles[i]->pos()));
            scene()->addItem(m_handles[i]);                    // ручице се убацују у сцену
          }
        }
        setPercent(100);
        m_onScene = true;
        emit statusText(tr("Use square handles to rotate. Use X handle to move the ..."));
        return true;
    }
    else
        return false;
}
```

Прилог 9. Метода за успостављање режима интеракције у класи AnimationRotation

При крају блока функције initializeForInteraction можемо видјети примјер како се емитује неки сигнал; с обзиром да је трансформација спремна да почне, емитује се информативни текст који објашњава како се користе ручице за дефинисање ротације.

Методе анимације које рукују догађајима који стижу са улазних уређаја углавном су једноставне. У случају анимација ротације, трансформације и транслације, ове методе користе уграђену имплементацију, да би дозволиле својим ручицама да се помјерају. У случају анимације за цртање новог објекта, сви догађаји се прослеђују том објекту, а он у зависности од врсте којој припада на овај начин препознаје своје параметре.

Графички објекти

За сваку врсту графичког објекта који желимо подржати, постоји по једна класа изведена из GraphicsObject. Та базна класа најбогатија је и најкомплекснија од досад наведених. Она садржи методе грубо подијељене на сљедеће групе:

- Реаговање на догађаје који дођу са улазних уређаја, обично прослијеђене из власничке анимације (миш, тастатура, контекстуални мени)
- Репродукција на основу XML чвора и стварање XML чвора на основу свог стања
- Трансформација параметара на основу другог објекта или садржаја графичких контрола
- Помоћне статичке функције

Приказаћемо скраћен приказ класе GraphicsObject, изузимајући оне методе и чланове класе који нису релевантни за приказ, у прилогу 10.

```
class GraphicsObject: public QGraphicsObject, public Name<GraphicsObject>
{
public:
    enum Type
    {
        PointType, LineType, TriangleType, RectangleType, EllipseType,
        CircleType, PolygonType, TextType, AngleType, NoType
    };
    GraphicsObject();
```

```
    GraphicsObject(const GraphicsObject &other);
    virtual ~GraphicsObject();

    static GraphicsObject* createObject(GraphicsObject::Type type);
    static GraphicsObject* getObjectFromXML(const QString &xml, const QString &nameAlt);

    virtual void mouseMoveEvent(const MousePosition &, const KeyModifiers &) = 0;
    virtual void mousePressEvent(QGraphicsSceneMouseEvent *event);
    virtual void mousePressEvent(const MousePosition &, const KeyModifiers &) = 0;
    virtual void mouseDoubleClickEvent(const MousePosition &, const KeyModifiers &);
    virtual void keyPressEvent(int key, const QString &ch);
    virtual void keyReleaseEvent(int key, const QString &ch);
    virtual void contextMenuEvent(QGraphicsSceneContextMenuEvent *event);

    virtual QString toHistoryString() const;
    virtual QWidget* getPaletteFrame();
    virtual void deletePaletteFrame();
    virtual QWidget* getObjectPaletteFrame() = 0;
    virtual void deleteObjectPaletteFrame() = 0;
    virtual bool snap(const QPointF &point, QPointF *snapPoint, qreal *distance) const;

    void transformTo(const GraphicsObject *object, qreal percent);
    virtual void translate(const QPointF &vec, qreal percent);

    QDomElement toXML(QDomDocument& root) const;
    bool recoverFromXML(const QDomElement&, QStringList&, bool, const QString &);
    QString toXMLString() const;

    virtual QVector<TransformationHandle*> getTransformationHandles() = 0;
    virtual QVector<RotationHandle*> getRotationHandles();
    virtual GraphicsObject *clone() const = 0;

protected:
    virtual QDomElement _toXML(QDomDocument& root) const = 0;
    virtual bool _recoverFromXML(const QDomElement& object, QStringList& errors) = 0;
    virtual void _transformTo(const GraphicsObject *object, qreal percent) = 0;
    virtual bool _objectSpecified(QStringList& errors) = 0;

public slots:
    bool objectSpecified();

protected:
    QPointF mapToParent(const QPointF &point) const;
    QPointF mapFromParent(const QPointF &point) const;
    QString m_name;
    QColor m_color;
    unsigned m_pointsDefined;
    qreal m_percent;
    Qt::PenStyle m_penStyle;
```

```
private:
   QFrame *m_frame;
   Ui::GraphicsObjectPalette *m_palette;

signals:
   void statusText(QString);
};
```

Прилог 10. Дефиниција класе GraphicsObject, базне
за све врсте графичких објеката

У класи GraphicsObject биће занимљиво позабавити се методама које рукују догађајима који стижу са улазних уређаја, прије свега кликом на лијево дугме миша. Клик на лијево дугме миша потребан је за цртање сваке врсте геометријског објекта. Један клик биће довољан за цртање тачке, два клика за цртање дужи и кружнице, три за елипсу, угао, правоугаоник и троугао и N кликова за многоугао. Уколико корисник зада више од потребног броја позиција на екрану него што је потребно за цртање неке врсте објекта, сувишни клик ће се третирати као први, односно цртање ће ићи „у круг“, док год корисник не саопшти да је објекат комплетиран. Да би се водила евиденција колико параметара је већ дефинисано кликом на дугме миша, у графичком објекту је декларисана промјењива под називом m_pointsDefined. Свака врста геометријског објекта интерпретираће ову промјењиву на себи својствен начин. У циљу представљања примјера, приказаћемо функцију mousePressEvent класе Rectangle у прилогу 11.

```
void Rectangle::mousePressEvent(const MousePosition &mPos, const KeyModifiers &km)
{
   prepareGeometryChange();

   int pd = getPointsDefined();

   if (pd % 3 == 0)
   {
      setTopLeft(QPointF(mPos .x(), mPos.y()));
      setA(0);
      setB(0);
      emit topLeftXChanged(QString::number( MY_ROUND(getTopLeft().x())));
      emit topLeftYChanged(QString::number(-MY_ROUND(getTopLeft().y())));
      emit statusText(tr("Now click once more to define the horizontal side, i.e. side A"));
   }
   else if (pd % 3 == 1)
   {
      setA(std::fabs(mousePosition.x() - getTopLeft().x()));
      emit AChanged(QString::number(MY_ROUND(getA())));
      emit statusText(tr("Now click once more to define the vertical side, i.e. side B"));
   }
   else
   {
      if (keyModifiers.shift())
         setB(getA());
```

```
      else
        setB(std::fabs(mousePosition.y() - getTopLeft().y()));
      emit BChanged(QString::number(MY_ROUND(getB())));
      emit statusText(tr("Rectangle defined. Press `complete` if you're ..."));
    }
    increasePointsDefined();
}
```

Прилог 11. Дефиниција методе Rectangle::mousePressEvent, која прихвата клик на дугме миша као један параметар у низу

Све врсте графичких објеката имају методу paint, која врши конкретно цртање објекта, тачку по тачку. Заправо, у нашем случају није неопходно рачунати сваку тачку, јер Qt подржава цртање свих објеката који су нам потребни, или простије објекте од којих композицијом можемо добити оне који су нам потребни. Метода Triangle::paint може се видјети у прилогу 12.

```
void Triangle::paint(QPainter *painter, const QStyleOptionGraphicsItem *, QWidget *)
{
  painter->setBrush(getBrush());

  if (getPercent() == 100)
  {
    if (getPointsDefined() > 0)
    {
      QPolygonF polygon;
      polygon.append(getA());
      polygon.append(getB());
      polygon.append(getC());
      painter->drawPolygon(polygon);
      if (getPointsDefined() % 3 == 0)
      {
        if (m_showNodes)
        {
          drawX(painter, getA());
          drawX(painter, getB());
          drawX(painter, getC());
        }
        if (m_showInnerCircleCenter)
          drawInnerCircleCenter(painter);
        if (m_showOutsideCircleCenter)
          drawOuterCircleCenter(painter);
        if (m_showGravityCenter)
          drawGravityCenter(painter);
      }
    }
  }
  else if (getPointsDefined() % 3 == 0)
  {
```

```
    QList<Line> lines = toLines();
    drawLinesPercent(lines, getPercent(), painter);
  }
}
```

Прилог 12. Дефиниција методе Triangle::paint, која обавља конкретно цртање објекта на сцени

Од сваког графичког објекта који ступи на сцену захтијеваће се да генерише сопствени скуп графичких контрола уз помоћ којих се могу дефинисати параметри тог објекта. У томе је графички објекат сличан анимацији, од које се захтијева то исто, а спојени приказ ових двају скупова графичких објеката биће постављен у врху прозора апликације, гдје га корисник може лако наћи. Метода чија имплементација треба да обезбиједи контроле за дефинисање параметара назива се getObjectPaletteFrame и дефинисана је у свакој поткласи класе GraphicsObject. Да се не би дефинисао изглед ових контрола из оквира изворног кода, што би било тешко и за имплементацију и за одржавање, оне су прво нацртане уз помоћ Qt Creator-a, а затим се динамички учитавају у оквиру ове методе, попуњавају тренутним параметрима и повезују са догађајима миша. Илустрација унапријед припремљених контрола, које служе за дефинисање параметара текстуалног објекта, може се видјети на слици 11.

Слика 11. Скуп контрола унапријед припремљених за уређивање параметара текстуалног графичког објекта – TextPalette

У прилогу 13 приказана је имплементација методе getObjectPaletteFrame за класу Triangle. У овом случају, структура која представља скуп контрола за уређивање параметара троугла носи име TrianglePalette. Видјећемо како се формира објекат овог типа и припрема за интеракцију.

```
QWidget *Triangle::getObjectPaletteFrame()
{
  // довољна је једна инстанца
  if (m_oFrame == NULL)
  {
    m_oFrame = new QFrame;
    m_oPalette = new Ui::TrianglePalette;
    m_oPalette->setupUi(m_objectFrame);
  }

  // постављају се вриједности у односу на тренутно стање
  m_oPalette->ax->setText(QString::number( getA().x()));
  m_oPalette->ay->setText(QString::number(-getA().y()));
  m_oPalette->bx->setText(QString::number( getB().x()));
  m_oPalette->by->setText(QString::number(-getB().y()));
  m_oPalette->cx->setText(QString::number( getC().x()));
```

```
    m_oPalette->cy->setText(QString::number(-getC().y()));

    // појединачне контроле се спајају с одговарајућим догађајима
    connect(this, SIGNAL(AXChanged(QString)), m_oPalette->ax, SLOT(setText(QString)));
    connect(this, SIGNAL(AYChanged(QString)), m_oPalette->ay, SLOT(setText(QString)));
    connect(this, SIGNAL(BXChanged(QString)), m_oPalette->bx, SLOT(setText(QString)));
    connect(this, SIGNAL(BYChanged(QString)), m_oPalette->by, SLOT(setText(QString)));
    connect(this, SIGNAL(CXChanged(QString)), m_oPalette->cx, SLOT(setText(QString)));
    connect(this, SIGNAL(CYChanged(QString)), m_oPalette->cy, SLOT(setText(QString)));

    // припремљен контејнер се враћа као резултат функције
    return m_objectFrame;
}
```

Прилог 13. Дефиниција методе Triangle::getObjectPaletteFrame

Снимање звука

За снимање звука неопходно је само упознати се са скупом класа које Qt обезбјеђује у ове сврхе, а тај скуп је малобројан и једноставан. Постоје двије основне класе за руковање догађајима који долазе са звучне картице: QAudioInput и QAudioOutput; прва служи за снимање звука, а друга служи за репродукцију звука. По мишљењу аутора овог рада, једини проблем с овим класама је то што захтијевају датотеку као извор, односно одредиште података. Уколико би дозвољавале снимање у меморијски низ (или бар меморијску датотеку), односно репродукцију из меморијског низа, поступак снимања или репродукције био би значајно једноставнији. Без таквих средстава, програмер је приморан да стално ствара и брише нове датотеке привременог карактера.

Сва програмска логика у вези са снимањем и репродукцијом звука смјештена је у класу RecordDialog. У прилогу 14. видимо скраћени приказ дефиниције ове класе.

```
class RecordDialog : public QDialog
{
public:
    RecordDialog(...);
    QString getSoundFilename() const;
    int getStartPosition() const;

protected:
    void startRecording();
    void startPlaying();

public slots:
    void start();
    void stop();
    void doAccept();
    void doReject();
    void newRecording();

protected slots:
```

```
    void updateTime();
    void stateChanged(QAudio::State);

private:
    Ui::RecordDialog *ui;
    QFile *m_inputFile, *m_outputFile;
    QAudioInput    *m_audioInput;
    QAudioOutput *m_audioOutput;
    bool m_recorded;
    QTimer *m_timer;
    int m_secondsElapsed;
    void resetTimer();
};
```

Прилог 14. Дефиниција класе RecordDialog, која се користи за интерактивно снимање и репродукцију звука

У дефиницији класе RecordDialog може се примијетити и употреба објекта QTimer, који нуди услуге штоперице, односно користимо је да бисмо мјерили и приказивали вријеме утрошено на снимање / репродукцију.

Када желимо да отпочнемо процес снимања, извршавамо сљедеће задатке:

1. Правимо нову датотеку или отварамо постојећу за потребе складиштења података
2. Налажемо жељени формат аудио снимка и испитујемо могућности система. Уколико није подржан жељени формат, тражимо од система да нам да формат најближи жељеном.
3. Правимо објекат класе QAudioInput, којем предајемо формат из корака 2.
4. Повезујемо догађаје које емитује објекат класе QAudioInput из корака 3. са својим слотовима, да бисмо могли реаговати на њих
5. Покрећемо снимање, позивом методе start на објекту из корака 3.
6. Покрећемо штоперицу, да бисмо мјерили вријеме

Приказаћемо наведене кораке у оквиру изворног кода. Имплементација методе RecordDialog::startRecording приказана је у прилогу 15.

```
void RecordDialog::startRecording()
{
    // корак 1.
    m_filename = getNewAudioFilename();
    m_outputFile = new QFile;
    m_outputFile->setFileName(m_filename);
    m_outputFile->open(QIODevice::WriteOnly | QIODevice::Truncate));
    // корак 2.
    QAudioFormat format;
    format.setFrequency(32000);
    format.setChannels(2);
    format.setSampleSize(16);
    format.setCodec("audio/pcm");
    format.setByteOrder(QAudioFormat::LittleEndian);
    format.setSampleType(QAudioFormat::SignedInt);
```

```
    QAudioDeviceInfo info = QAudioDeviceInfo::defaultInputDevice();
    if (!info.isFormatSupported(format))
        format = info.nearestFormat(format);
    // корак 3.
    m_audioInput = new QAudioInput(format, this);
    ui->record->setIcon(QIcon(":/audio/images/audio/recording.png"));
    ui->record->setEnabled(false);
    ui->stop->setEnabled(true);
    // корак 4.
    connect(m_audioInput, SIGNAL(stateChanged(QAudio::State)), this,
SLOT(stateChanged(QAudio::State)));
    // корак 5.
    m_audioInput->start(m_outputFile);
    // корак 6.
    resetTimer();
    m_timer = new QTimer;
    m_timer->setInterval(1000);
    connect(m_timer, SIGNAL(timeout()), this, SLOT(updateTime()));
    m_timer->start();
}
```

Прилог 15. Поступак снимања звука у оквиру класе RecordDialog

Подешавања

У основи подршке за подешавања у оквиру библиотеке Qt лежи класа QSettings. Ова класа подржава датотеке у INI формату, али и формате датотека за подешавања која зависе од окружења оперативног система. Мајкрософт виндоуз има подршку за смјештање подешавања у системском регистру, а у линуксу корисничка подешавања најчешће су смјештена у скривеним директоријумима у матичном директоријуму. Да би се превазишла оваква разноликост, довољно је прослиједити параметар конструктору класе QSettings, који јединствено представља наш пројекат и којим од њега захтијевамо да испита окружење на којем је покренут програм и обезбиједи нам приступ корисничким подешавањима наше апликације. Уколико она нису постојала, биће направљена, а уколико су постојала, добићемо њихово стање и биће нам омогућено да их мијењамо. Програмски код који руководи овим догађајима смјештен је на самом почетку главне функције:

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QSettings *settings = NULL;
    settings = new QSettings(QSettings::UserScope, "JeckaSoft", "Homework");
    Settings::getInstance()->initialize(settings);
    ...
    return a.exec();
}
```

У оквиру класе Settings користиће се методе класе QSettings да би се прибавили или уписали подаци, а преостали код ове класе служи за графички приказ, валидацију

података и интеракцију са корисником.

– – – – – – – – –

Изворни код за другу компоненту система, прегледач, нећемо наводити у овом раду. Овдје смо жељели приказати основне карактеристике имплементације и приказати њене кључне компоненте. Већина класа овдје описаних користи се и у прегледачу – менаџер сцене, класе за анимације, графичке објекте, репродукцију звука, итд. Неке од њих су прилагођене (нпр. менаџер сцене и анимације), а неке су коришћене на идентичан начин као у дизајнеру (нпр. графички објекти). Цјелокупан код је доступан у прилогу овог рада.

~~Верификација~~Обезбјеђење квалитета

Comment [Д.М.1]: Верификација ј име фазе у моделу Водопад, па сам се трудио да наслови одражавају имена појединих фаза, али што се мене тиче може и овако, јер је јасније и неће довести до двосмислености.

У случају комерцијалног производа ~~верификација~~ обезбјеђење квалитета заузима скоро исти временски период и уложени труд као и сам развој. Тимови за тестирање софтвера раде у тијесној вези за развојним тимом, у току развоја, као и након развоја главне цјелине производа. Критеријуми функционалности постављени су већ у раној фази, програмери су се трудили да их задовоље, а тимови за тестирање пажљивом анализом откривају грешке и предају их развојном тиму на поправку. У случају овог рада, спроведена је детаљна ~~верификација~~ провјера задовољења~~у односу на~~ првобитно ~~постављене~~ постављених ~~циљеве~~циљева, уклоњене су бројне грешке (багови) које су се појавиле у току имплементације и преостале су неке суптилне грешке које ће бити размотрене у каснијим фазама. За потребе овог рада биће довољно илустровати корисност пројекта на реалном примјеру.

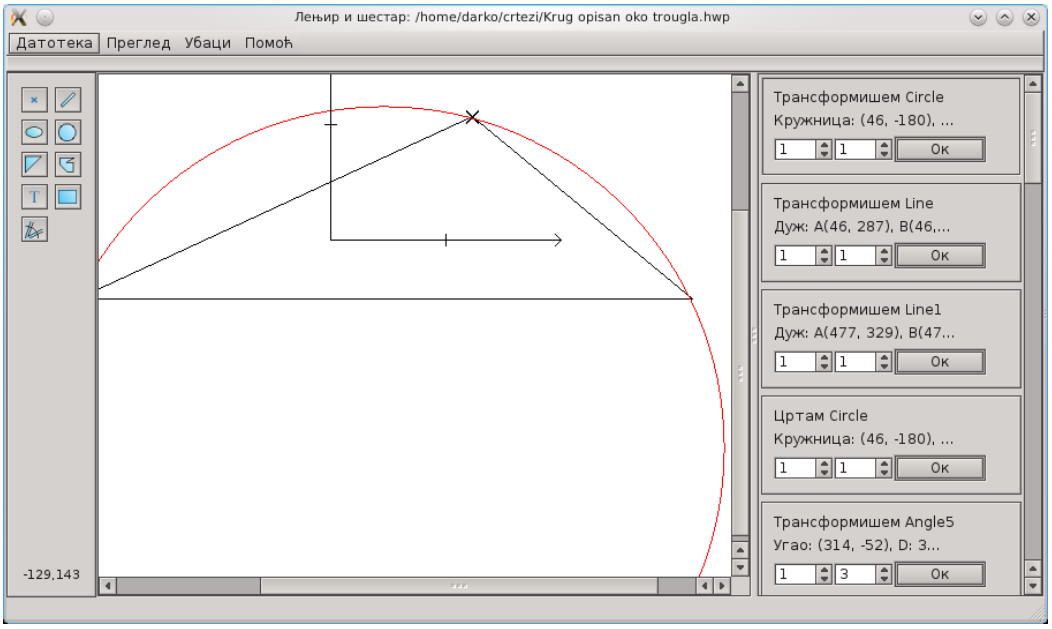
Примјер 2. Описати конструкцију кружнице описане око троугла.

Рјешење: Процес ћемо описати корак по корак:

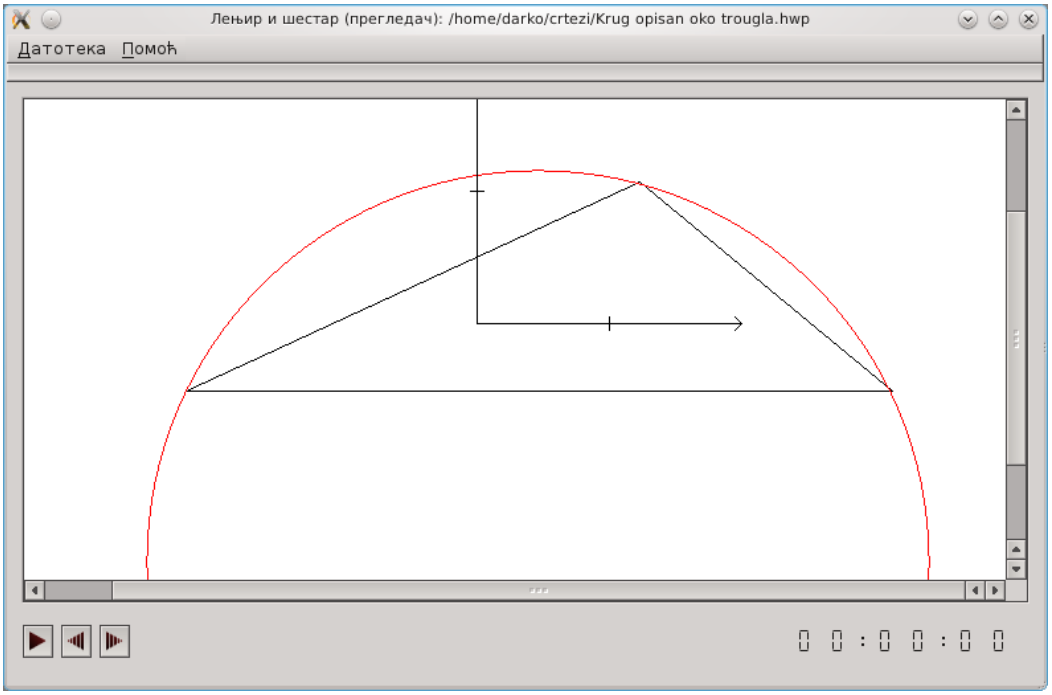
1. Отварамо главни прозор дизајнера и бирамо алат „троугао“
2. Кликнемо по једном на три различита положаја на екрану да бисмо добили троугао ABC
3. Кликнемо „готово“ да бисмо означили да смо задовољни датим троуглом
4. Бирамо алат „лук“ и цртамо по два лука истог пречника са центрима у A и B, тако да се пресијецају (након цртања сваког лука, кликнемо на дугме „готово“)
5. Бирамо алат „дуж“ и цртамо дуж MN која спаја пресјеке лукова из корака 5.
6. Понављамо корак 4 за тачке B и C и корак 5 да бисмо добили дуж OP
7. Бирамо алат „круг“
8. Цртамо круг са центром на пресјеку дужи MN и OP
9. Снимамо датотеку

Добијену датотеку можемо пустити уз помоћ прегледача. У циљу једноставности, овај случај нисмо пропратили гласовним коментарима.

Резултати конструкције у дизајнеру и прегледачу приказани су на сликама 12 и 13.



Слика 12. Кружница описана око троугла, конструисана и приказана у дизајнеру



Слика 13. Кружница описана око троугла, приказана у прегледачу

Закључак

Овим радом покушано је указати ~~покушали смо указати~~ на могуће мјере по питању давања информатичке подршке школству, у контексту наставе математике. Заправо, у покушају да се да илустрација развоја једног таквог пројекта, уједно ~~смо~~ су ~~приказали~~ приказани ~~неке~~ неки од основних корака које треба предузети да би се организовао и остварио софтверски пројекат, почев од анализе захтјева, идући преко одабира технологија и развоја архитектуре, до конкретних корака имплементације у развојном окружењу и изворном коду. На тај начин развијен је комплетан програм који може представљати основу даљег развоја, да би у једном тренутку могао представљати оно за шта је намијењен – за унапређење наставе математике кроз самостални ваннаставни рад ученика. Даљи развој могао би обухватати бољу подршку за звук (нпр. увоз готових датотека), олакшан кориснички интерфејс, више пречица на тастатури, памћење параметара посљедњег нацртаног објекта да би се употријебили у наредном, поновно извршење посљедњег корака итд.

Пројекат који ~~смо описали~~ је описан има врло узак дјелокруг. ~~Изабрали смо~~ Изабрана је основно- школска планиметрија ~~школску планиметрију~~ у којој се већи дио градива своди на учење различитих поступака конструкције или израчунавања, те су подобни за приказ у форми анимације. Друге области математике захтијевале би посебан софтвер, а свакако да посебан изазов представља развој пројеката за подршку другим школским предметима.

Носиоци развоја пројеката везаних за школско градиво у идеалном случају били би сами наставници, који су детаљно упућени у План и програм, имају искуство у раду са дјецом и евидентно су опредијељени за наставни рад, што треба да значи да су обогаћени ентузијазмом за сопствени развој и развој наставе. Истовремено, рад наставника у овом домену није предвиђен Планом и програмом и било би неопходно користити слободно вријеме за развој, а финансијски статус наставника у Србији није задовољавајући. Зато је неопходно да се значај тих пројеката препозна, како у Министарству просвете Републике Србије, тако и у ширим круговима нашег друштва, те да се пружи одговарајући подстрек њиховом развоју.

Литература

1. Стратегија развоја образовања у Србији до 2020. године (СРОС), 2012. Министарство просвете и науке Републике Србије
2. Јовановић, Небојша. *Како се наштимовати уз помоћ фидбека*, 2005. ИСБН 86-83797-31-7
3. План и програм наставе математике за основну школу Министарства образовања Републике Србије, 2011.
4. GeoGebra Conference Report, 2010. http://www.geogebra.org/conferences/2009-GeoGebra-Conference-Linz.pdf
5. Марић, Милена. *Представљање математичког садржаја на интернету*, мастер рад, 2011.
6. Херцег, Драгослав; Херцег, Ђорђе. *GeoGebra – динамичка геометрија и алгебра*, 2007. Нови Сад
7. Britannica, The New Encyclopaedia, 2007.
8. Royce, Winston W. *Managing the Development of Large Software Systems*, 1970.