Izrada obrazovnog softvera za poučavanje nastavnih sadržaja iz elektromagnetizma

Čačić, Adrijan

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Department of Physics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za fiziku**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:160:227565

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ODJEL ZA FIZIKU

ADRIJAN ČAČIĆ

IZRADA OBRAZOVNOG SOFTVERA ZA POUČAVANJE NASTAVNIH SADRŽAJA IZ ELEKTROMAGNETIZMA

Diplomski rad

Osijek, 2017.

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ODJEL ZA FIZIKU

ADRIJAN ČAČIĆ

IZRADA OBRAZOVNOG SOFTVERA ZA POUČAVANJE NASTAVNIH SADRŽAJA IZ ELEKTROMAGNETIZMA

Diplomski rad

predložen Odjelu za fiziku Sveučilišta J. J. Strossmayera u Osijeku
radi stjecanja zvanja Magistra edukacije fizike i informatike

Osijek, 2017.

Ovaj diplomski rad je izrađen u Osijeku pod vodstvom izv.prof.dr.sc. Vanje Radolića u sklopu Sveučilišnog diplomskog studija Fizike i informatike – nastavnički smjer na Odjelu za fiziku Sveučilišta Josipa Jurja Strossmayera u Osijeku.

Sveučilište J.J.Strossmayera u Osijeku
Odjel za fiziku

Diplomski rad

IZRADA OBRAZOVNOG SOFTVERA ZA POUČAVANJE NASTAVNIH SADRŽAJA IZ ELEKTROMAGNETIZMA

ADRIJAN ČAČIĆ

Sažetak:

U radu se detaljno pojašnjava postupak izrade softvera uz prethodni naglasak o mogućnostima njegove uporabe u nastavi. U prvom dijelu rada je rasprava o ulozi računala u nastavi kao i učeničkim pretkoncepcijama u području elektromagnetizma s ciljem prepoznavanja potrebe za izradom softvera za nastavu kako bi se one uspješnije otklonile. Nakon toga slijedi kratko upoznavanje alata za izradu softvera i konceptualna razrada algoritma za simuliranje strujnog kruga. Glavni dio rada čini detaljna uputa za izradu softvera uz dodatke izvornog programskog koda u svrhu reprodukcije softvera za korištenje ili proširenje osnovnog programa. U završnom dijelu rada priložena je priprema za nastavni sat održan uz pomoć izrađenog softvera, uz primjer prilagodbe softvera specifičnoj jedinici gradiva.

Rad je pohranjen u knjižnici Odjela za fiziku
Ključne riječi: elektromagnetizam / fizika / izrada / nastava / softver
Mentor: izv.prof.dr.sc. Vanja Radolić
Ocjenjivači: doc. dr. sc. Marina Poje Sovilj
mr. sc. Slavko Petrinšak
Rad prihvaćen: 21. prosinca 2017.

J. J. Strossmayer University in Osijek
Department of Physics

Bachelor of Science Thesis

DEVELOPMENT OF EDUCATIONAL SOFTWARE FOR TEACHING ELECTROMAGNETISM

ADRIJAN ČAČIĆ

Abstract:

The thesis discusses the process of development of the software for teaching electromagnetism and its application in physics education. A brief discussion on the most common students misconceptions on electromagnetism is followed after the explanation of the purpose of computer implementation in education. Furthermore, a short introduction on the use of software tools in development of the educational software along with an elaboration of the algorithm used to simulate an electric circuit is given. The software development instructions of the educational software are the main part of the thesis with the further insight into the possible changes and adaptions of the software to fulfil requirements and needs of the teachers and students in class. The final part consists of a teachers preparation document as an example of the educational software being used in class.

Thesis deposited in Department of Physics library
Keywords: education / electromagnetism / physics / software development
Supervisor: Vanja Radolić, PhD, Associate Professor
Reviewers: Marina Poje Sovilj, PhD, Assistant Professor
Slavko Petrinšak, MSc
Thesis accepted: 21st of December, 2017.

Sadržaj:

1. Uvod

Jedan od rijetko spomenutih poslova svakog učitelja vječna je potraga za novim tehnikama ili metodama prenošenja znanja svojim učenicima. Zahvaljujući napretku tehnologije, potraga te krajnje pronalaženje istih danas je, više nego ikada, uvelike olakšana. Unatoč tome, nastavne materijale na koje nailazimo širokog su spektra kvalitete i primjene, te nisu pogodni za korištenje na nastavi. Neki od najčešćih su: neprilagođenost uzrastu učenika, materijali na stranom jeziku, razlike u znakovlju i nazivu fizikalnih veličina i pojava, preširoko ili preusko pokriveno područje i slično.

Kao budući nastavnik fizike i informatike vidim odličnu priliku u projektima samostalne izrade softvera kojim se možemo služiti u nastavi fizike ili informatike, a kojima ujedno prakticiramo i unaprjeđujemo znanje i fizike i informatike.

Elektromagnetizam je područje fizike koje učenicima često stvara probleme u razumijevanju temeljnih koncepata zbog njihove apstraktnosti kao i, pomanjkanja opreme za izvođenje pokusa. Softver koji simulira strujni krug početna je zamisao ovog rada i jedno od rješenja već spomenutih problema koji omogućuje učenicima samostalno eksperimentiranje tijekom i izvan nastave. Izrada softvera opisana je poglavljima i opis prati gotove cjeline rada, umjesto tijeka izrade koji je po prirodi fragmentiran i težak za pratiti. Potpuni programski kod je dostupan i zapisan u tekstualnim okvirima po poglavljima izrade. Praćenjem uputa moguće je samostalno stvoriti kopiju softvera te dalje nadograđivati izrađeno. Za kraj rada pripremljen je primjer školskog sata nastave fizike u osnovnoj školi pri kojem se softver koristi u nastavi.

Detaljnim opisom postupka izrade softvera i dijeljenjem gotovog projekta s kolegama nadam se približiti konceptualne probleme fizike algoritamskom razmišljanju informatike i računalstva te možda ohrabriti sljedećeg nastavnika za samostalni pokušaj izrade sličnog materijala. Cilj je nadahnuti istomišljenike na rad i time stvoriti osnovu interaktivnih digitalnih alata u svrhu interaktivnije i svrsishodnije nastave fizike i informatike.

2. Računalo u nastavi fizike

Nastava fizike bitan je dio obrazovanja učenika kojim budimo zanimanje za okolinu u kojoj se učenik nalazi, potičemo istraživačko razmišljanje te prenosimo modele razmišljanja temeljene na eksperimentu i znanosti. U današnjem tehnološkom dobu, gdje napredak nije u koraku s generacijskom smjenom, bitno je poticati duh znanosti i cjeloživotnog učenja primjenom novih tehnologija u nastavi. Računalo u nastavi nije novi koncept i primijenjen je u modernoj nastavi sa širokim spektrom uspješnosti. Od jednostavne zamjene školske ploče kao prezentacijskim oruđem do potpune integracije interaktivnih alata u nastavi. Velika dostupnost multimedijskih i interaktivnih materijala daje veliki izbor u obogaćivanju nastave fizike kako u motivacijskom tako i u konceptualnom dijelu.

Tijekom nastave fizike veliku prednost predmeta predstavlja prikazivanje svakodnevnih pojava, predmeta i događaja, koje učenici uzimaju zdravo za gotovo, kao motivaciju. Rušenjem pretkoncepcija i predviđanjem slučajeva koje dosad nisu iskusili, učenike se potiče da razmišljaju o svijetu u kojem žive te isti vide u novom, informiranom svjetlu. Pokusi, demonstracije i općenito praktične vježbe daju priliku učeniku stvoriti opipljivo iskustvo zakonitosti koje proučava te pomaže u razvoju vještina i razumijevanju fundamentalnih koncepata fizike, a time i prirode koja ga okružuje. Idealno bi iste koristili pri svakom dostupnom satu nastave, što praktično nije izvedivo iz mnogih razloga. Manjak dostupne ili prikladne opreme, nedostatak prostora i vremena, višak ili manjak učenika sve su poteškoće zbog kojih pokuse i demonstracije stavljamo u drugi plan i time osiromašujemo učeničko iskustvo nastave fizike.

Dostupnost računalne i komunikacijske tehnologije nudi alternativu. Korištenjem interaktivnih sadržaja u nastavi potiče se rad u kojem učenici imaju petlju rada i povratne informacije. Stvara se sigurna okolina u kojoj mogu eksperimentirati sa zadanim parametrima i time poduprijeti obrađenu teoriju praktičnim radom. Učenje kroz interaktivnu nastavu potiče aktivan proces učenja gdje se naglašava suradnja učenika međusobno i suradnja učenika i učitelja. Interaktivni sadržaji omogućuju i određenu razinu samoinicijative kojom učenici testiraju svoje hipoteze i ideje bez socijalnog pritiska ili straha od fizičkih posljedica. Interaktivna nastava i računalni materijali oruđe je kojim nastavu fizike treba nadopuniti ili obogatiti, a ne trik kojem pribjegavamo iz potrebe skretanja pažnje.

Određena područja nastave fizike nisu prikladna za pokuse ili je iste teško izvesti. Sloboda koja dolazi prikazom računalnom simulacijom zaobilazi te probleme i mogu se stvoriti simulacije bez straha od opasnosti ili nepredvidivih posljedica. Računalna okolina interaktivnih sadržaja nudi prednosti koje dolaze s računalima u općem smislu. Mogućnosti upravljanja okolinom, varijablama

i parametrima, vizualnim pokazateljima i informacijama, izbor izmjene, pohrane i prilagodbi potrebama. Na drugoj strani stoje i manjak opipljivosti računalnih simulacija u kontrastu s iskustvom pravog svijeta. Nije potrebno simulirati nešto što lako možemo pokazati učenicima pred njihovim očima.

3. Pretkoncepcije u elektromagnetizmu

Principi elektromagnetizma neizbježni su u današnjem svijetu koji počiva na tehnologiji i tehnološkom razvoju. Unatoč stalnom kontaktu s električnim pojavama i primjenama, nastava fizike iz područja elektromagnetizma problematična je mnogim učenicima. Stalan kontakt s tehnologijom nije nužno ključ razumijevanja elektromagnetskih pojava i služi nastajanju raznih pretkoncepcija. Osim opće kontraintuitivnosti fizike, problem elektromagnetizma nastaje manjkom vidljivih posljedica električnih i magnetskih pojava. Učenička iskustva staju s posljedicama korištenja električnih pojava u svrhu brzog i učinkovitog prijenosa te korištenja energije. Potrebno je, stoga, učeničkim idejama i pretkoncepcijama prići s oprezom, pokušati shvatiti začetak krivog shvaćanja te sustavno ispravljati iste.

Mnoga istraživanja razine konceptualnog shvaćanja fizike učenika raznih razina obrazovanja provedena su i objavljena, a njihovi rezultati i zaključci doveli su do sistematizacije najčešćih pretkoncepcija učenika i strategija za njihovo otklanjanje. U kontekstu rada, potrebno je istražiti prekoncepcije elektromagnetizma.

Električna energija pojam je koji učenici ne povezuju s konceptima energije iz drugih područja fizike. Poučeni svakodnevnim iskustvima „trošenja električne struje“ zanemaruju principe očuvanja. Također, energiju fizički predstavljaju masom ili volumenom izvora energije.

Električna struja je „štaka“ na koju se učenici oslanjaju pri opisivanju pojmova i pojava elektromagnetizma, koristeći ga u svakodnevici na taj način. Veoma često problemi nastaju pri objašnjenju toka struje griješeći pri stvaranju strujnog kruga bez shvaćanja uloge vodiča i polova izvora.

Za napon učenici čuju prethodno, ali izbjegavaju izraz ili ga poistovjećuju s električnom strujom. Osim sinonima, napon često preuzme i ulogu struje u strujnom krugu što uvelike otežava kasnija objašnjenja potencijala.

Pri razumijevanju otpora pretkoncepcije nastaju u objašnjenju međusobnog utjecaja otpora, napona i struje. Otpor je često svojstvo samog trošila ili otpornika. Spajanjem više otpornika u različitim konfiguracijama ne utječe na rezultat ukupnog otpora kruga ili otpora jedne grane kruga. Objašnjavanje i spajanje strujnih krugova također predstavlja izvor problema u nastavi fizike. Nerazlikovanje paralelnih od serijskih spojeva te ponašanje struje i napona u svakom dijelu. Nemogućnost predviđanja karakteristika određene grane ili ponašanja električnih komponenti s obzirom na ostatak kruga. Mjerne instrumente učenici vide kao nezavisne čimbenike koji nemaju utjecaja na strujni krug.

Problem veoma bitan u kontekstu ovog rada je spajanje jednostavnog strujnog kruga i objašnjenje toka struje općenito. Pruži li se učenicima okolina za eksperimentiranje s izvorom, trošilom i vodičima moguće je vidjeti razne ideje, greške pa i objašnjenja pri spajanju elemenata. Pretkoncepcije je korisno svrstati u najčešće kategorije:

1. Jednopolni model – spajanje izvora i trošila s jednim vodičem pri čemu struja teče od izvora prema trošilu.
2. Sudarački model – točno spojen izvor i trošilo pomoću dva vodiča pri čemu je smjer struje zamišljen kao sudar koji kreće iz oba pola izvora.
3. Potrošački model – model u kojem je objašnjenje električne energije tvar koja se „troši“ pri nailasku na trošilo pri čemu se nepotrošeni ostatak vraća u izvor.
4. Model dijeljenja – gdje se struja u ganjanju dijeli podjednako čime je objašnjeno svjetlo žaruljica u paralelnim spojevima

Svaki model predstavlja set izazova sam po sebi, ako se uzme u obzir pokušaj usvajanja daljnjih koncepata toka struje, napona, otpora itd. jasno je zamisliti manjak motivacije i razumijevanja koji doživljavaju učenici u samom početku elektromagnetizma. Mogućnost proširenja jednostavnog strujnog kruga dodatnim elementima i tehnikama spajanja istih otežava problematiku još više. S obzirom na tematiku ovog rada potrebno je istražiti probleme na koje učenici nailaze spajajući, prikazujući i interpretirajući strujne krugove, njihove sheme i očitane vrijednosti. Ponovno je prepoznat obrazac najčešćih pretkoncepcija pri interpretaciji strujnih krugova koji su kategorizirani u sljedećem obliku:

1. Izvor-potrošač model – već spomenuti model proizlazi iz krivog ili manjka shvaćanja temeljnih pojmova poput napona, električne energije, struje i napona. Struja je resurs koji se djelomično troši pri nailasku na trošilo te se ostatak vraća u izvor.

2. Sekvencijalni model – gdje učenik smatra da promjena nastala negdje u strujnom krugu utječe samo na sljedeće ili sekvencijalne elemente strujnog kruga, bez posljedica na prethodne.

3. Lokalni model – slično prethodnom modelu, očituje se manjkom razumijevanja utjecaja promjena u strujnom krugu na cijeli strujni krug. Promjene u intenzitetu svjetla žaruljice dodavanjem otpornika u paralelu s žaruljicom objašnjavaju tendencijom struje da prolazi „ondje gdje joj je lakše“.

4. Model neovisnih grana – slično prethodnim modelima, greške nastaju pri pokušaju objašnjenja promjena i utjecaja jedne grane na druge grane u strujnom krugu. Najčešće promjene jedne grane utječu samo na tu granu.

5. Model realnih shema – učenici često shemu strujnog kruga poistovjećuju sa stvarnim slikom strujnog kruga i njegovih elemenata. Problemi nastaju pri fizičkim odnosima veličina (dimenzije elemenata, duljine vodiča, raspored elemenata) i apstraktnim prikazima spojeva (npr. višestruko grananje koje je, ovisno o izvedbi vodiča, teže postići u stvarnom spoju).

6. Model neovisnih mjernih instrumenata – ampermetar i voltmetar u ovom modelu služe samo za mjerenje napona i struje bez ikakvog utjecaja na sami strujni krug. Problem je lako otkriti upitom u razlog za spajanje ampermetra izričito serijski, a voltmetra paralelno u strujni krug.

Korištenjem računalne simulacije moguće je postaviti okruženje u kojem će se postupno, na vizualni način, voditi učenika u otkrivanje problema njihove interpretacije strujnog kruga. Također se stvara okruženje u kojem se učenika poziva na slobodno eksperimentiranje, bez negativnih posljedica i opasnosti. Kreiranje specifičnih problema za testiranje učeničkih pretpostavki ili pretkoncepcija također je odlika simulacije strujnog kruga, koja brzo i izravno omogućava suočavanje s problemom te osigurava brzu ispravku početne pogrešne ideje zbog koje je nastao sukob između iskustva i predstavljenih činjenica. Dostupnost softvera na raznim uređajima, brze povratne informacije i problemske situacije „skrojene“ učeničkim pretkoncepcijama omogućuju učiteljima interaktivnu nastavu koja potiče eksperimente i zaključke.

4. Izrada softvera

Prije početka rada potrebno je odlučiti područje, cjelinu ili jedinicu gradiva fizike koju želimo obraditi kako bi se problem rastavio na manje dijelove koje je time lakše predstaviti algoritmima. Svako područje fizike predstavit će jedinstvene izazove pri apstrakciji i predočenju programskim jezicima. Potrebno je odrediti platforme, s obzirom na potrebe učenika ili učitelja, na kojima želimo da softver bude dostupan. Također bitno je odlučiti hoće li simulacija biti prikazana dvodimenzionalno ili trodimenzionalno, pušta li se zvuk i hoće li biti dostupan na mjestu izvođenja vježbe, načine upravljanja programom, veličina zaslona ili prezentacijske površine itd.

Ovaj softver smišljen je za korištenje u učionici prikazan projektorom, dakle velikom slikom lošije oštrine, i upravljan mišem na računalu. Alternativno, postoje i druge mogućnosti ovisno o opremljenosti učionice i učenika. Program je vrlo lako prilagoditi za korištenje s dodirnom površinama poput onih na tabletu, pametnoj ploči i *smartphone* uređajima. Na taj način se može učenicima pružiti vlastita inačice programa u kojem mogu samostalno raditi, čak i van nastave.

Prikaz korisničkog sučelja, kontrast i kompoziciju elemenata, treba uzeti u obzir pri izradi i vjerojatnost prikazivanja na velikim zaslonima razlog su za odabir jednostavne estetike crnih elemenata na svijetloj pozadini.

4.1. Odabir programskog paketa za izradu softvera

Na današnjem softverskom tržištu dostupni su mnogi paketi softvera koji se mogu iskoristiti za izradu multimedije, filma, animacije, zvuka, glazbe itd., i interaktivnog medija - računalne igre i digitalne simulacije. Izrada bilo kojeg spomenutog medija nije više ograničena na stručnjake niti je zatvorena iza skupog softvera i manjka materijala za formalno ili samostalno obrazovanje. Uz malo volje i vremena, te uz pomoć interneta, danas više nego ikada, moguće je samostalno izraditi softver.

Jedan od velikih izazova nastavnika, učitelja i profesora je pronalazak i izrada materijala koje koriste pri nastavi. Materijali koje često skupo plaćaju osobno ili pomoću školske ustanove. Samostalan pronalazak nastavnih materijala često daje neučinkovite rezultate. Materijali su često nepotpuni, zaključani iza novčane cijene kroz reklamu ili u obliku promotivnog uzorka. Neprecizni ili nekompatibilni, specifično za predmet fizike, materijali koriste različite standarde mjernih jedinica, simboli i oznake, fizikalne veličine, teme i jedinice nastave fizike nisu jednake ovisno o zemlji porijekla nastavnih materijala. Moguće rješenje je samostalna izrada vlastitih nastavnih

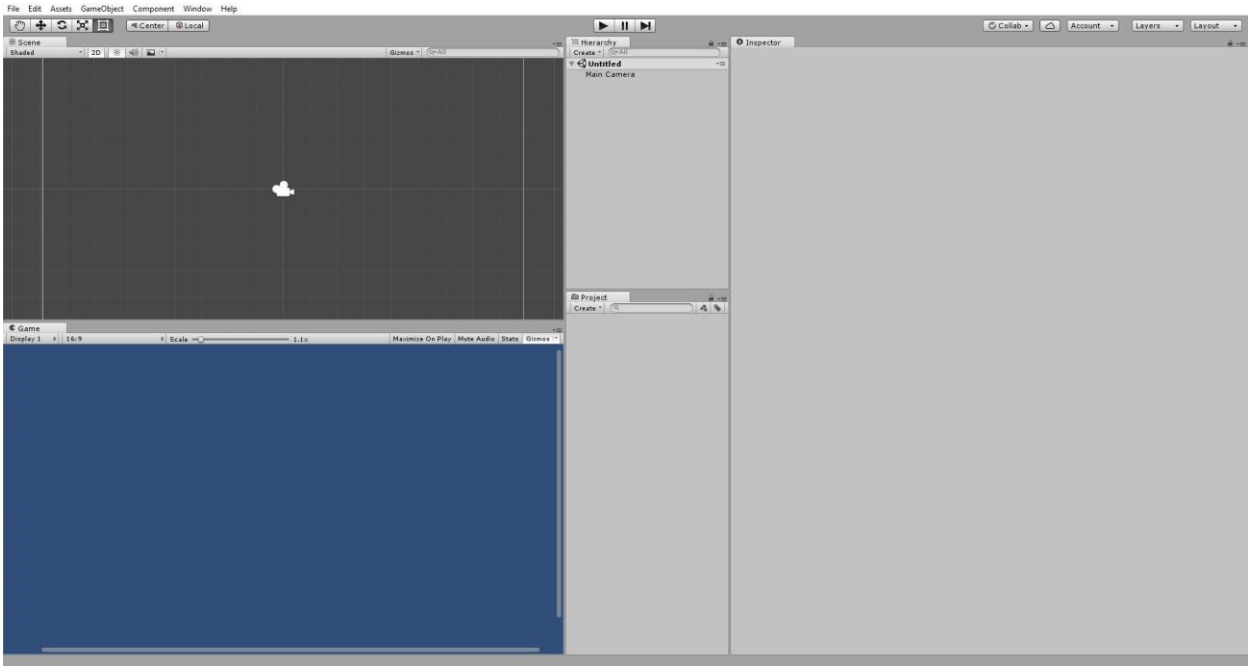
materijala koje je zatim lako održavati ili unaprijediti ovisno o potrebama ili promjenama plana i programa nastave fizike.

Unity3D ili samo *Unity*, softverski je paket tvrtke *Unity Technologies*[1] za izradu računalnih igara, animacija i aplikacija. Velika prednost *Unity* razvojnog okruženja je novčana dostupnost. Osobna licenca je besplatna u slučaju da pomoću nje zarađujete manje od sto tisuća američkih dolara, ili valutni ekvivalent, godišnje. Plaćena profesionalna licenca daje pogodnosti u obliku korisničke podrške, dodatne mogućnosti uređivanja postavki programa, primjerice, besplatna inačica uvijek pri početku rada izrađenog programa ima logo tvrtke *Unity Technologies* dok plaćena inačica omogućava proizvoljno uređivanje. U sklopu podrške programski paket ima trgovinu na kojoj korisnici postavljaju izrađene objekte i programski kod za prodaju i korištenje. Popularnost i jednostavnost korištenja uvjetovale su raširenost materijala za učenje i rad s programom *Unity*. Pregršt tečajeva, lekcija, koncepata i gotovih projekata, profesionalnih i amaterskih, plaćenih i besplatnih dostupni su na najprije pri pretraživanju interneta, ali i fizički u obliku knjiga i u sklopu učilišta. Projekt izrađen u programu *Unity* moguće je zapakirati u izvršnu datoteku za mnoge platforme kao što su *Windows, MacOS* i *Linux* kao velike platforme računalnih operativnih sustava, *Android* i *iOs* iz kuta mobilnih platformi te razne web platforme kao što je popularni *HTML5*.

4.2. Osnove korištenja programa *Unity*

U daljnjem dijelu rada izložen je princip izrade softvera korištenjem programa *Unity* koji se kao niz uputa može koristiti za samostalnu izradu istoga softvera. Za razumijevanje određenih koraka potrebno je objasniti osnove korištenja programa Unity. Sučelje programa sastoji se od nekolicine osnovnih prozora (Slika 1.) koji obavljaju specifične funkcije u radu s programom. To su: osnovna alatna traka *Toolbar* (sadrži izbornike za manipulaciju projekta i datoteka te pristup svim ostalim prozorima i mogućnostima programa), prozor scena *Scene view* (prikazuje dio projekta kojim se trenutno manipulira), prozor igre *Game view* (prikazuje što trenutno vidi odabrana kamera), prozor hijerarhije *Hierarchy* (prikazuje popis svih objekata koji se koriste u sceni koje je trenutno manipulirana), prozor projekta *Project* (popis je datoteka i svih podataka koji se koriste u izradi projekta), prozor preglednika *Inspector* (prikazuje svojstva odabranog objekta ili podatka), *Visual Studio 2017* - razvojno okruženje tvrtke Microsoft, koje se u suradnji koristi za pregled i pisanje programskog koda.

[1] Sve informacije vezane za Unity3D paket podložne su promjenama i ovdje predstavljaju stanje rujna 2017. godine



Slika 1. Glavno sučelje programa Unity.

4.2.1. Alatna traka

Osim uobičajenih izbornika pomoću kojih se učitavaju, čuvaju i općenito obrađuju podaci (rezanje, dupliciranje, umnožavanje), alatna traka sadrži i izbornike pomoću kojih se dolazi do najčešće obavljenih radnji pri stvaranju objekata i prikazu određenih prozora radne okoline. Upravljanje ovim dijelom alatne trake je jednostavno i imitira rad većine programa neovisno o operativnom sustavu. Ispod osnovnog dijela alatne trake nalazi se traka koja je rjeđa u svojoj učestalosti (Slika 2.). Pet ikona s lijeve strane pomoćne trake odabiru metodu upravljanja objektima u prozoru scene, koji se po zadanom nalazi ispod istih.

Slika 2. Ikone za upravljanje objektima u prostoru.

Ikona otvorene ruke služi za pomicanje pogleda prozora po površini okomitoj na kut gledanja, uz pomoć kotačića miš mijenja se udaljenost od same površine, odnosno udaljava se i približava pogled prozora. Sljedeća ikona služi za pomicanje odabranog objekta. Obojane strjelice prikazuju individualne osi po kojima se pomiče objekt, alternativno se objekt može i slobodno pomicati pomoću plavog kvadrata pokraj sjecišta osi. Nadalje, ikona s dvije zakrivljene strjelice služi za rotiranje trenutno odabranog objekta. Oko odabranog objekta nastane sfera s obojenim kružnicama. Obojene kružnice ograničavaju rotaciju oko osi koja se prepoznaje bojom, dok sama sfera omogućava slobodnu rotaciju ovisnu o kutu gledanja prozora scene. Analogno prethodnim

ikonama, ikona prikazana strjelicama koje odlaze iz kutova kvadrata, povlačenje obojenih pravaca povećava ili smanjuje objekt u smjeru odabrane osi, a središnja kocka omogućava slobodno upravljanje veličine objekta u prostoru. Povećanje objekta djeluje množenjem dimenzija objekta[2], poveća li se objekt dvostruko, njegove dimenzije (visina, duljina i širina) ostaju iste, ali mijenja se koeficijent uvećanja određene osi. Zadnja ikona s lijeve strane pomoćne alatne trake služi za izravnu promjenu dimenzija određenog objekta. Sljedeće dvije ikone, koje su prostorno odijeljene od prethodnih, pomažu postavljanju perspektive objekata pri njihovom uređivanju. Ikona *Center* omogućava uređivanje središta odabranog objekta te služi određivanju relativnosti položaja u pogledu na ostale objekte. Sljedeća ikona nudi izbor uređivanja parametara objekata u lokalnom i globalnom pogledu, odnosno manipuliranju objekta u kontekstu cijele scene ili drugih objekata.

U središtu pomoćne trake nalaze se tri ikone koje su zaslužne za testno pokretanje programa, pauziranje trenutnog rada programa i skok od jedne sličice iscrtavanja osnovne petlje programa.

Slika 3. Administrativne kartice.

Desna strana pomoćne trake odgovorna je za izbornike (Slika 3.) korisničkog računa, postavki rasporeda prozora *Unityja*, *Layout*, te slojeva *Layers* koji služe za organizaciju slojeva prikaza objekata na sceni, u slučaju više slojeva koji se trebaju preklopiti.

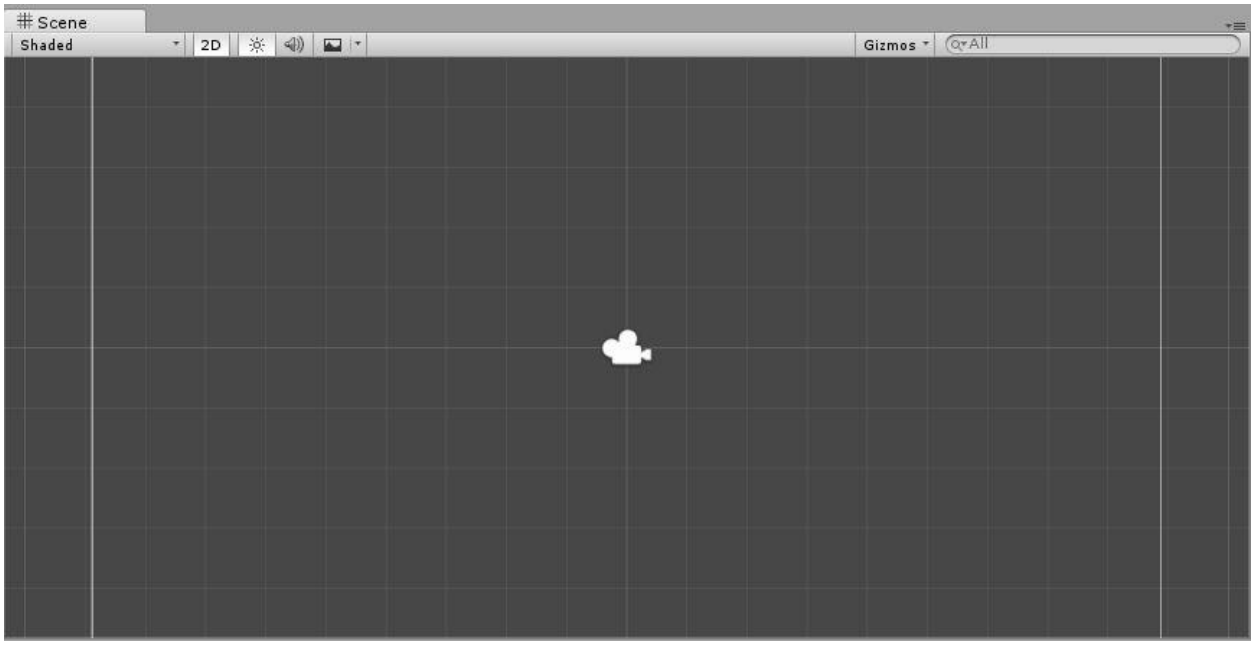
4.2.2. Prozori scene i igre

Scena u *Unityju* ima posebnu ulogu ugrađenu u sami softverski paket. Prvotna zadaća programa *Unity* kao programskog paketa za izradu računalnih igara, koju je u međuvremenu nadišao, vidi se u mnogim aspektima korisničkog sučelja kao što su imena prozora *Game* i *Scene*. Scene su tvorci programa zamislili kao zasebne dijelove programa koje je potrebno posebno učitati kako bi se koristili. Često se projekt stoga sastoji od glavne, početne scene[3] u obliku izbornika i ostalih scena koje se učitavaju po potrebi, poput različitih razina u računalnim igrama. U slučaju da je projekt manjeg djelokruga jedna scena može biti sasvim dovoljna, poput programa izrađenog kroz ovaj rad. Snaga scena može se implementirati u rad ukoliko autor želi, primjerice, unaprijed pripremiti različite konfiguracije strujnih krugova ili u jednom programu izraditi simulacije više područja fizike. Jedna scena koja simulira strujni krug, druga u kojoj se simulira međudjelovanje magneta, treća u kojoj otpornost vodiča o fizikalnim karakteristikama itd.

[2] *Scale* je koeficijent koji nam olakšava pravilno uvećanje objekata, često se izbjegava zbog razvlačenja grafičkih vrijednosti objekata

[3] Početna scena stvara se automatski ulazom u novi projekt

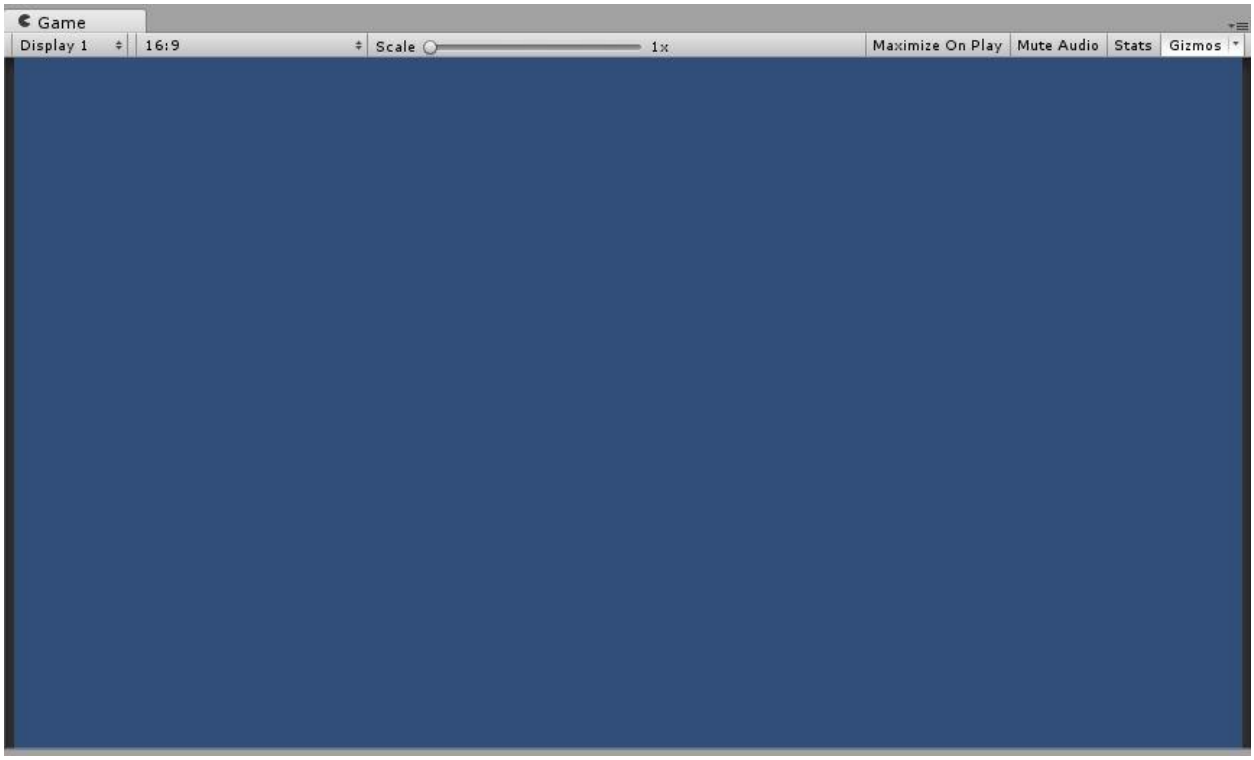
Poput svakog prozora u *Unityju,* lijevim klikom i povlačenjem na karticu *Scene* može se premjestiti sami prozor na željeno mjesto. Kao što je već spomenuto, prozor scene (Slika 4.) prikazuje sve elemente trenutne scene. Unutar ovog prozora je moguće manipulirati (translatirati, rotirati, uvećati, umanjiti) svakim objektom u trenutnoj sceni (svim objektima koji se trenutno nalaze na popisu hijerarhije). Izravno upravljanje objektima moguće je samo u prozoru scene. Prozor igre prikazuje prvi pogled na ono što će korisnik vidjeti kada pokrene scenu programa i izvorne postavke prikazuju pogled glavne kamere. Služi za vizualnu referencu pri uređivanju objekata i u njemu nije moguće manipulirati objektima u prostoru. Oba prozora imaju pomoćne alatne trake pri svom vrhu.



Slika 4. Prozor scene.

S lijeve na desno, pomoćna traka prozora scene najprije daje izbor prikaza sjenčanja i tekstura svih objekata u trenutnoj sceni. Primjer korisnosti ovog izbornika vidljiv je ukoliko se treba provjeriti sijeku li se dva objekta fizički, isključi se sjenčanje objekata te ostavi najjednostavniji prikaz žičanog okvira, *wireframe*[4]. Nadalje, sljedeći gumb služi za prebacivanje između 2D i 3D sučelja. 2D sučelje postavi z-os okomito na površinu gledanja, time os postaje os dubine. U klasičnom 2D programu os-z nema funkciju jer se svi objekti nalaze na istoj udaljenosti od glavne kamere. Pri korištenju 3D sučelja u desnom kutu prozora stoji osno pomagalo koje bojama i svojom rotacijom daje do znanja kako je vidno polje prozora orijentirano. Sljedeća tri gumba služe, slično prvom izborniku, za paljenje i gašenje objekata određenih kategorija, kao što su zvuk, svjetlost i razni efekti kamere.

[4] Prikazuje bridove poligona od kojih se sastoji svaki trodimenzionalni objekt moderne grafike



Slika 5. Prozor igre.

Pomoćna traka prozora igre (Slika 5.) najprije nudi izbornike za odabir zaslona za prikaz i promjenu razlučivosti zaslona, najčešće korišteni za provjeru izgleda programa na različitim platformama. Klizač u središnjoj poziciji prozora ima jednostavnu ulogu uvećanja slike radi provjere i ugađanja vizualnih elemenata programa. Gumbi na desnoj strani prozora najprije daju izbor povećanja prozora pri pokretanju programa i gašenju zvuka. Zatim, u slučaju problema ili optimizacije, koristi se informativni prozor za prikaz hardverske statistike tokom vrtnje programa. Izbornik *Gizmos*, poput istog izbornika u prozoru scena, omogućava prikaz raznih vizualnih pomagala, npr. osi pri uređivanju objekata.

4.2.3. Prozor hijerarhije

Prozor hijerarhije (Slika 6.) predstavlja popis svih objekata trenutne scene te njihov međusobni odnos. Objekti mogu biti zasebni ili ugniježđeni u grane klasičnom vezom roditelj-dijete. Prije pokretanja programa, svi objekti postavljeni u sceni bit će prikazani u prozoru hijerarhije. Međutim nakon pokretanja programa, ovisno o programskom kodu koji se izvršava, moguće je uživo promijeniti stanje hijerarhije dodavanjem i uklanjanjem objekata. Najjednostavnije je gledati na hijerarhiju kao na spremnik trenutno odabrane scene. Ukoliko, primjerice, program tijekom vremena usporava računalo, preporučljivo je provjeriti stvara li se velika količina objekata u popisu hijerarhije bez njihovog uklanjanja. Pomoćna alatna traka prozora ima najprije izbornik za stvaranje

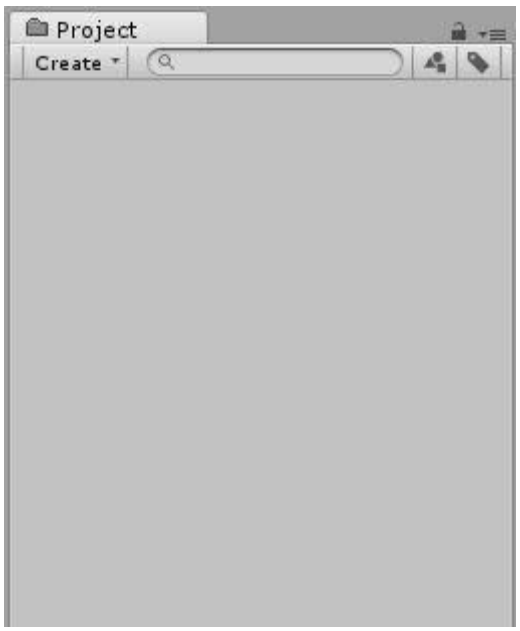
predložaka objekata koji se nalaze u samom programu *Unity* potpuno jednak izborniku *Gameobject* glavne alatne trake ili izbornika koji nastaje desnim klikom miša, i unosne trake za pretragu hijerarhije prema imenu objekta ili zadane mu oznake. Svaka nova scena sadrži objekt *Main Camera* kao početni objekt.

Slika 6. Prozor hijerarhije.

4.2.4. Prozor projekta

Prozor projekta (Slika 7.) služi za organizaciju projekta ili programa na kojem se radi na način da se podaci istog tipa drže u mapama. Na taj način lako se nalaze željeni podaci i pomoću izbornika se može prikazati kako su podaci različitih tipova međusobno povezani. Pomoćna alatna traka prozora jednaka je alatnoj traci prozora hijerarhije te dodatno omogućava stvaranje datoteka i promjenu naziva podataka. Vrste podataka koje najčešće sadrži prozor su: slike, zvukovi, teksture, 3d modeli, sve različite scene programa, skriptne datoteke[5] programskog koda te predloške objekata. Preporučljivo je, pri početku rada na novom projektu, stvoriti mape koje će sadržavati tipove datoteka kao što su skripte programskog koda, vizualne sadržaje, scene i predloške u svrhu organizacije.

[5] Datoteke koje sadrže zapisani kod odabranog programskog jezika

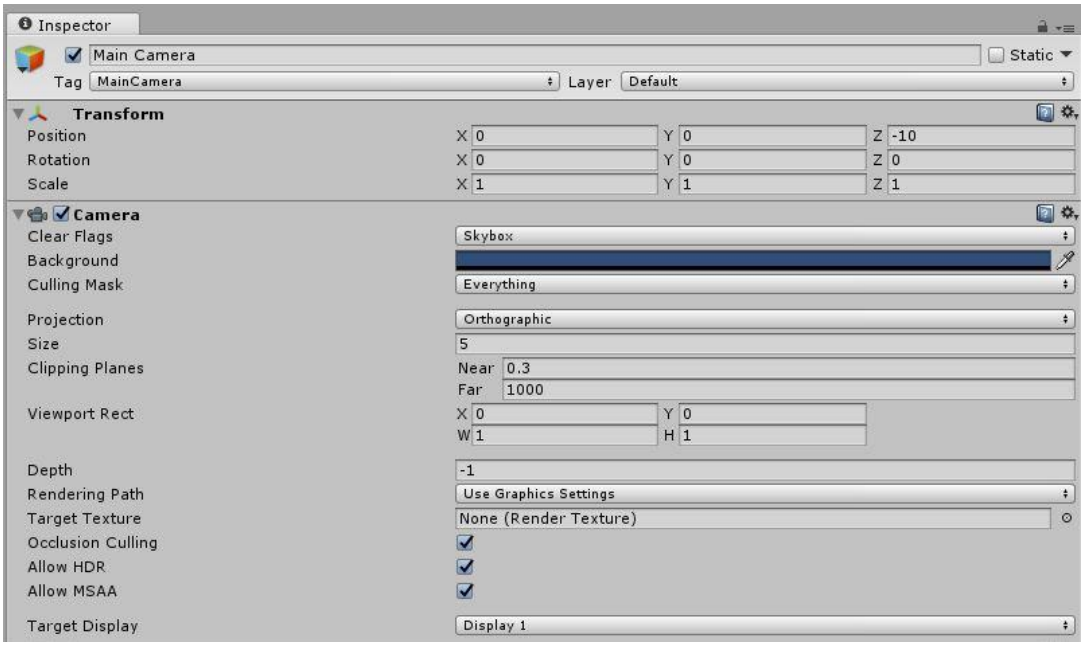


Slika 7. Prozor projekta.

4.2.5. Prozor preglednika

Prozor preglednika ili *Ispector* (Slika 8.) najmoćniji je prozor u kontekstu mogućnosti promjena parametara objekata, pa čak i samog programa *Unity*. Objekti se sastoje od komponenti, svaka komponenta samostalan je dio programskog koda koji se u pregledniku očituje kao kartica i zaseban dio prozora. Svaku komponentu moguće je isključiti i urediti.

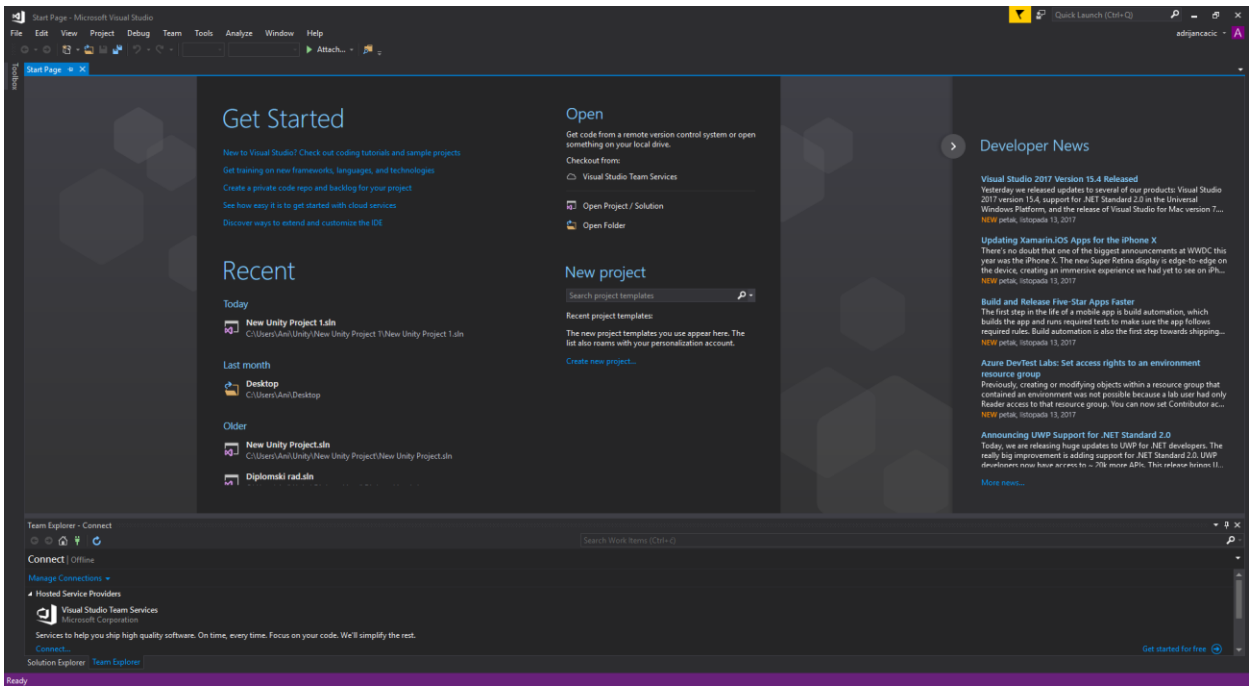
Ovisno o vrsti komponente, sadrže prozore, kliznike, grafove, gumbe i slike kojima se uređuju razni parametri objekata. Svaki pojedini ugrađeni objekt u programu ima vlastite unaprijed određene kartice u pregledniku koje sadrže zajedničke mogućnosti drugih objekata ili posebne, specifične mogućnosti odabranog objekta. Primjerice, velika većina objekata ima komponentu pod nazivom *transform* koja služi za definiciju i uređivanje položaja, rotaciju i mjerilo objekta. Izravnim uređivanjem brojčanih vrijednosti ili jednostavnim povlačenjem pomoću miša moguće je precizno, višestruko i probno uređivanje objekta. Velika moć programa je organizirano upravljanje veličinama, varijablama i mogućnostima kako bi postigli zamišljenu funkcionalnost softvera.



Slika 8. Prozor preglednika s odabranim objektom glavne kamere.

4.2.6. Visual studio 2017

Pisanje programskog koda moguće je bilo kojim urednikom teksta koji se pokreće otvaranjem skripte programskog koda u projektnom prozoru. *Unity* od nedavno dolazi u instalacijskom paketu s Microsoftovim *Visual studio 2007* razvojnim okruženjem. Paket je moguće koristiti besplatno obaveznom prijavom i registracijom Microsoftovim računom. Prijašnje inačice programa *Unity* dolazile su s besplatnim *Monodevelop* paketom iste funkcionalnosti. Prednosti uređivanja programskog koda pomoću ovih programa ovise o stilu pisanja programskog koda i osobnim izborima. Programi čitaju programski kod i označavaju prepoznate dijelove bojama, podcrtavaju greške i predlažu ispravke, poravnavaju i uređuju programski kod za preglednost itd. Pisanje programskog koda za *Unity* moguće je pomoću dva programska jezika: C# programski jezik, pogodan onima s iskustvom u izradi aplikacija na *Windows* platformi i programski jezik *Javascript* onima s iskustvom u razvoju web aplikacija. Program u ovom djelu izrađen je u *C#* programskom jeziku.



Slika 9. Početni prozor razvojnog okruženja Visual Studio 2017.

4.2.7. Osnove C# programskog jezika

C#[6] programski jezik je objektno orijentirani programski jezik u kojem su osnova objekti koji međusobno komuniciraju postupcima nazvanim metodama ili funkcijama. Jezik sadrži mnoge rječice pomoću kojih se daje do znanja programskom prevoditelju o kojoj vrsti objekta ili postupka je riječ. Rječice su veoma često prikazane bojama kako bi se programski kod lakše čitao, isto se vidi u tekstualnim kutijama unutar rada.

Rječica *using* prva je rječica koju je moguće susresti u skriptnim datotekama ovog rada. Koristi se za učitavanje gotovih, ugrađenih knjižnica metoda i funkcija koje dolaze s određenim programom, u ovom slučaju *Unity*. Rječica *class* služi za deklaraciju tipova podataka ili držača koji nisu predviđeni programskim jezikom, na taj način mogu se stvoriti vlastite podatkovne strukture koje je moguće mijenjati i vezati metodama i funkcijama. Klase, metode, funkcije itd. moraju nositi unikatno ime koje se unosi pazeći na pravila velikih slova i zabranjenih znakova. Naposljetku, ključne riječi dolaze u mnogo oblika i slažu se uz već spomenute dijelove programskog jezika. Primjer je ključna riječ *public* pomoću koje se daje ostatku programa do znanja da je varijabla ili klasa javna i time vidljiva svakom dijelu programa, potpuna suprotnost jest ključna riječ *private* koja varijablu zatvara svim ostalim dijelovima programa i time sprječava neželjene promjene. Izvršni redovi programskog koda moraju završavati znakom točke zareza (;), dok blokovi

[6] čita se u obliku: *C sharp* , na engleskom jeziku

programskog koda imaju graničnike u obliku vitičastih zagrada, otvorenih na početku i zatvorenih na kraju bloka. Nedostatak jednog od tih znakova najčešća je greška pri pisanju programskog koda.

4.3. Algoritam strujnog kruga

Prvotna zamisao ovog rada, izraditi nekolicinu unaprijed određenih strujnih krugova konstruiranih za specifične probleme, ambiciozno je narasla na izradu sustava za dinamičnu izradu i upravljanje strujnih krugova u multimedijskom okruženju. Mogućnosti rada s programom uvelike su se povećale, omogućivši korisnicima izradu i istraživanje mnoštva drugih konfiguracija strujnih krugova.

Izazov izrade programa također je narastao te više nije bilo dovoljno iskoristiti samo vizualne alate *Unity* okruženja te ih popratiti specifičnim programskim kodom, prikladnom problemu koji se prikazuje. Nastala je potreba za izradom logike strujnog kruga, koja prati otpore elemenata strujnog kruga, napon izvora te dinamično izračunava struju svake grane strujnog kruga koristeći pri tome odgovarajuće matematičke modele. Problem nastaje pri pokušaju definiranja paralelnog i serijskog spoja otpornika i izvora. S obzirom da se ne može predvidjeti konfiguracija i broj paralelno i serijski spojenih elemenata koje korisnik želi spojiti, potrebno je izraditi algoritam koji će samostalno očitati i provesti račun strujnog kruga. Svaki element je potrebno zapisati, posložiti s obzirom na ostale elemente te, poznajući ulogu svakog elementa, odrediti jakost struje, pad napona i otpor svakog elementa. Osnovni elementi u strujnom krugu su izvor kao unos napona, trošilo kao nosač otpora, sklopka kao sustav provjere i kontrole stanja kruga i vodič uloge poveznika ostalih elemenata strujnog kruga.

Logiku strujnog kruga potrebno je zapisati u zasebnim dijelovima povezanima komunikacijom. Jedan veliki problem kod pisanja logike bit će prikaz grananja strujnog kruga, koji je prikladno prikazati matricama zbog neodređenog broja razina. Matrični prikaz povlači za sobom odgovornost matričnog računa koji nije nipošto lako prikazati jezikom programskog koda. Logika kruga najveći je izazov ovog rada.

4.4. Izrada sučelja

Nakon pokretanja programa *Unity,* novi projekt se stvara u početnom prozoru odabirom tipke *New*. U sljedećem prozoru unosi se ime projekta i njegova lokacija na tvrdom disku. S desne strane prozora ponuđen je izbor postavke za 2D i 3D okruženje koja postavlja prozor igre i scene u

prikladnu konfiguraciju i time omogućuje ili uklanja z-os prostora. Za potrebe izrade ovog softvera odabire se 2D, jer se shema strujnog kruga prikazuje u jednoj ravnini, te kliknuvši na gumb *Create Project* program ulazi u prikaz glavnog sučelja (Slika 1.).

4.4.1. Kamera i osvjetljenje

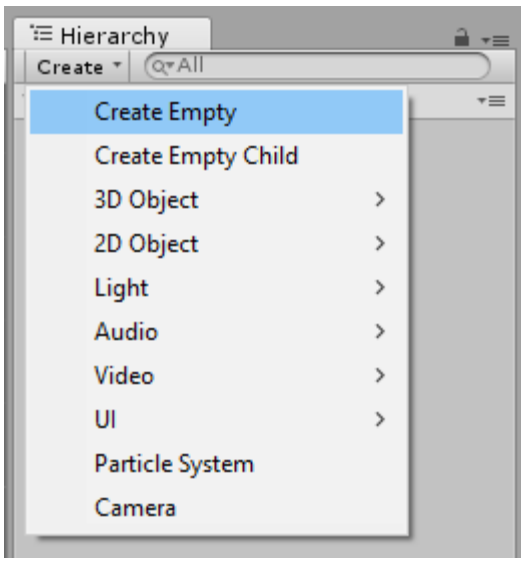
Svaki novostvoreni projekt u programu Unity započinje s istim objektom u hijerarhiji i sceni, glavnom kamerom nazvanom *Main camera*. Glavna kamera daje vidno polje programa u trodimenzionalnom ili dvodimenzionalnom prostoru programa. Poput svakog objekta u programu, moguće je manipulirati kamerom pomoću skripte, mijenjati kut, uvećanje, poziciju kamere itd. Za potrebe ovog rada kamera ostaje statična i pogled na prostor radi ortografski, u slučaju ako je odabrana dvodimenzionalna postavka pri izradi novog projekta. U slučaju da glavna kamera nema ortografsku postavku, isto se može učiniti u inspektor prozoru nakon što se u hijerarhiji glavna kamera označi lijevim klikom. Kartica *Camera* pod retkom *Projection* za odabir ima meni gdje se označi *Ortographic*[7].

S obzirom na način iscrtavanja prostora u programu, potrebno je imati i osvjetljenje prisutno. Trodimenzionalni objekti iscrtavaju se pomoću tekstura i sjenčanja. Bez svjetla objekti poprimaju veoma mračan izgled i njihova boja ne dolazi do izražaja. Problem se rješava dodavanjem predloška objekta klikom na meni *GameObject* u alatnoj traci, zatim se označi *Light* (Slika 10.) i klikne na *Directional light* u pomoćnom izborniku. Novonastali *Directional light* označi se u hijerarhiji klikom te u inspektoru pod karticom *Transform* rotacija se postavi na nulu s obzirom na sve tri osi, skala se ostavlja na jedan, a pozicija se postavlja nula s obzirom na x i y osi dok se z os postavlja na -10. Time se svjetlo postavlja iza glavne kamere i izravno osvjetljava sve vidljive objekte. Promjenom kuta usmjerenja svjetla mogu se postići različiti efekti sjene.

4.4.2. Izbornik

Prazan objekt stvara se u sceni klikom na *GameObject* te *Create empty* (Slika 10.), objekt se može preimenovati u izbornik, desnim klikom na objekt u hijerarhiji i odabirom *Rename* u padajućem izborniku. Objekt izbornik služi kao nositelj skripte, koja će ujediniti objekte koji se dinamično stvaraju tijekom korištenja programa te vizualno učitava elemente strujnog kruga. S obzirom da nema vizualnih komponenti njegova pozicija na sceni nema utjecaja ostavi li se kao početnu ili ako se vrati na ishodište prostora.

[7] Način iscrtavanja 3D slike na 2D plohu



Slika 10. Stvaranje gotovih objekata.

U prozoru *Project*, u slučaju da ne postoji, prazna mapa se može stvoriti desnim klikom bilo gdje u prozoru i odabirom *Create folder* te ju preimenovati u skripte. U ovoj mapi nalazit će se sve skriptne datoteke koje su stvorene tijekom rada. Desnim klikom na mapu te odabirom *Create c# script* stvara se nova skripta pod imenom *Newbehaviourscript* koju je odmah potrebno preimenovati u izbornik. Unutar ove skripte unosi se programski kod kojim se predstavlja rad izbornika. Bitno je napomenuti da novostvorene skripte nastaju s unaprijed upisanim programskim kodom koji služi kao osnova za pisanje daljnjeg programskog koda, u slučaju ovoga rada programski kod je sigurno obrisati i krenuti ispočetka. Lako je moguće da programski kod koji se upiše, pogotovo na početku rada, daje greške u konzoli programa *Unity* nakon što se sačuvaju skripte i program provjeri sve napisano. Postoje mehanizmi koji pregledaju pozivanje funkcija i reference na druge objekte u skriptama. S obzirom da u početnim koracima stvaranja softvera nedostaje objekata i drugih skripti koje se pozivaju ovom prvom skriptom, program će upozoriti na isto i onemogućiti testiranje programa dok pogreške ne budu ispravljene. Sintaksa C# programskog jezika nije u potpunosti objašnjena ovim radom, određeni dijelovi bit će obrađeni radi lakšeg shvaćanja i mogućnosti kasnije izmjene ili nadogradnje. Dvoklikom na skriptu izbornik otvara se skripta u programu za uređivanje skripti te se upisuje programski kod izložen u tekstualnom prozoru ispod. U bilo kojem trenutku pisanje skripte moguće je sačuvati i skriptu povući na objekt izbornika u prozoru hijerarhije. Učini li se to vidljivo je da je objekt izbornika dobio novu karticu u prozoru *inspector*.

Objekt izbornika služit će za pohranu i pozivanje elementa strujnog kruga. Prva linije skripte unutar klase izbornika bit će definicije varijabli i objekata koje će pomoći pri tome. U nastavku programskog koda definirat će se funkcije kojima se pozivaju unaprijed izgrađeni elementi kruga, pri tom ih se učitava pohranom u varijable oblika *GameObject.* Ostale varijable služe za pohranu i

određivanje početnog napona izvora i otpora otpornika koji se stvaraju klikom na gumbe izbornika. Pomoću varijabli v*rijednostNapona* i v*rijednostOtpora* mogu se postaviti druge početne vrijednosti koje su vidljive na gumbima izbornika. Funkcija *OnGUI* ugrađena je u *Unity* i pokreće se po učitavanju elemenata korisničkog sučelja, odnosno odmah nakon pokretanja programa u ovom slučaju. Unutar funkcije provjerava se jesu li zadane vrijednosti napona i otpora unutar dopuštenih granica. U nastavku, funkcije *StvoriIzvor, StvoriVodic, StvoriOtpornik* i *StvoriPrekidac* čine upravo ono što im imena govore, stvaraju inačicu elemenata kruga i, u slučaju otpornika i izvora, šalju izvještaj o stvorenom elementu funkcijama koje brinu o ustrojstvu kruga obrađene dalje u radu. Ove funkcije pozivaju se klikom na gumbe izbornika čija je izrada obrađena u sljedećem paragrafu.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class Izbornik : MonoBehaviour {

       public GameObject gotoviVodic;
       public GameObject gotoviIzvor;
       public GameObject gotoviOtpornik;
       public GameObject gotovaSklopka;
       public string vrijednostNapona = "1.5";
       public string vrijednostOtpora = "10";
       private float napon = 0.0f;
       private float otpor = 1.0f;
    public InputField inputNapon;
    public InputField inputOtpor;

    void OnGUI()
    {
        vrijednostNapona = inputNapon.text;
        if (float.TryParse(vrijednostNapona, out napon))
        {napon = Mathf.Clamp(napon, 0.0f, 10.0f);}
        else
        {napon = 0.0f;}

        vrijednostOtpora = inputOtpor.text;
        if (float.TryParse(vrijednostOtpora, out otpor))
        {otpor = Mathf.Clamp(otpor, 1.0f, 10000.0f);}
        else
        {otpor = 1.0f;}
    }
    public void StvoriIzvor ()
    {
        GameObject obj = Instantiate(gotoviIzvor) as GameObject;
        IzvorSpoj battery = obj.GetComponent<IzvorSpoj>();
        if (battery != null)
            battery.napon = napon;}

    public void StvoriVodic()
    {
        Instantiate(gotoviVodic);}
    public void StvoriOtpornik()
    {
        GameObject obj = Instantiate(gotoviOtpornik) as GameObject;
        OtpornikSpoj rNode = obj.GetComponent<OtpornikSpoj>();
        if (rNode != null)
        {
            rNode.otpor = otpor;}
    }
    public void StvoriPrekidac()
    {Instantiate(gotovaSklopka);}}
```

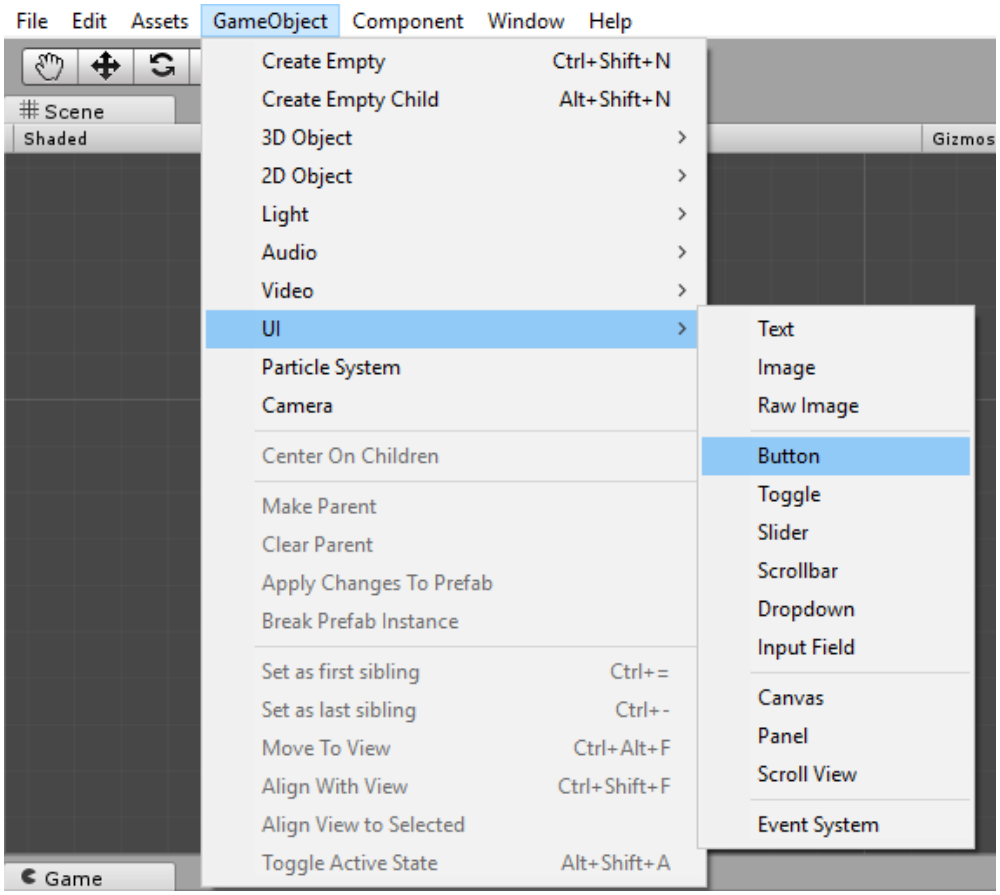
Skriptna datoteka 1. Izbornik.

4.4.3. Canvas

Iskoristivši relativno nove mogućnosti programa Unity vizualni dio izbornika programa izrađen je pomoću ugrađenih *User Interface*[8] komponenti. Stvaranjem bilo koje UI komponente iz izbornika *GameObject* na alatnoj traci, u hijerarhiji nastaje objekt nazvan *Canvas* uz koji vidimo i strjelicu s lijeve strane. *Canvas* predstavlja grafičko sučelje trenutne kamere i uvijek će biti roditelj objekt svim UI objektima. Veza roditelj-potomak objekata pomaže u organizaciji, nasljedstvu skripti programskog koda te manipulaciji više objekata odjednom. U ovom slučaju *Canvas* se postavlja kao dvodimenzionalno platno ispred svih drugih objekata koje je zalijepljeno za glavnu kameru. U slučaju pomicanja kamere u prostoru, elementi sučelja ostaju uvijek jednako udaljeni i orijentirani. Osim *Canvas* objekta nastaje i *EventSystem* objekt koji obavlja ulogu vršenja skriptnog programskog koda povezanog s UI elementima. Nije nužan za rad već nudi alternativno vizualno sučelje za interakciju između UI elemenata i ostalih objekata; sve to pa i više moguće je postići skriptnim programskim kodom.

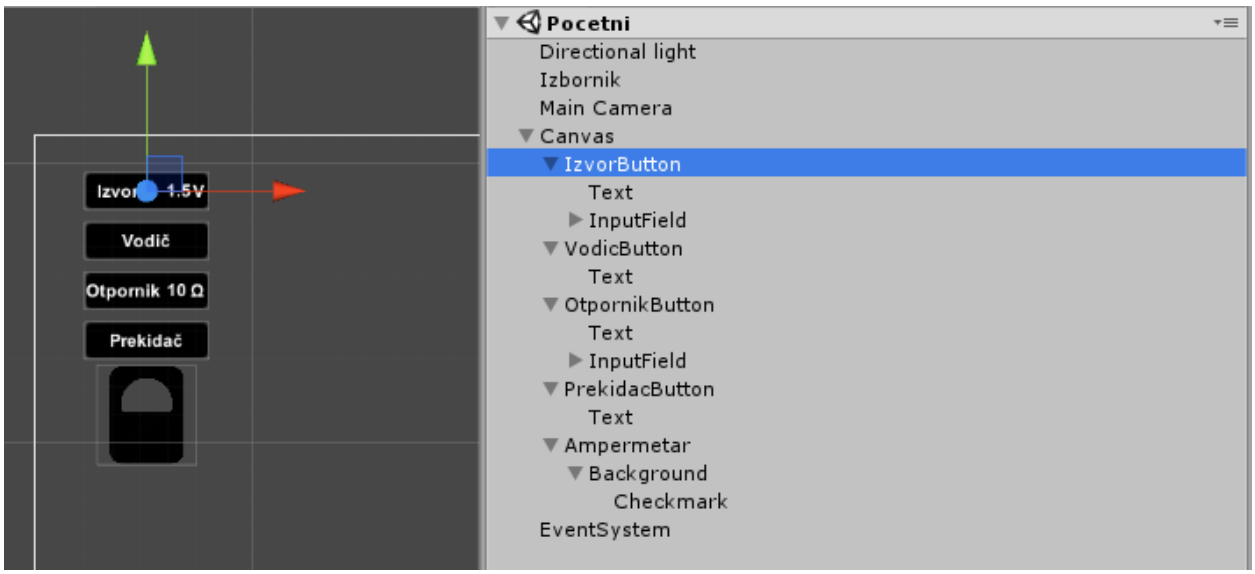
Sučelje programa zamišljeno je u obliku izbornika sastavljenog od gumbova postavljenog s lijeve strane vidnog polja, ispod zadnjeg gumba dodan je vizualni element nazvan ampermetar koji služi kao obavijest korisniku da strujnim krugom teče struja. Vizualni stil strujnog kruga veoma je blizak shematskom prikazu strujnog kruga kako bi vizualno podsjećao učenike na naučeno, s jednostavnim crnim pravcima u obliku shematskih prikaza. Također, sastavljene krugove moguće je slikati i lako postaviti kao strujni krug u drugom mediju, primjerice dokumentu ili prezentaciji. Izrada vizualnog prikaza naboja u pokretu nije predstavljena u ovom radu zbog mogućnosti otežanog razumijevanja od strane učenika, dodatno spoj vizualno gubi na jasnoći i, u većim krugovima, količina elemenata u pokretu može usporiti slabije uređaje.

[8] Skraćeno UI



Slika 11. Stvaranje UI objekata.

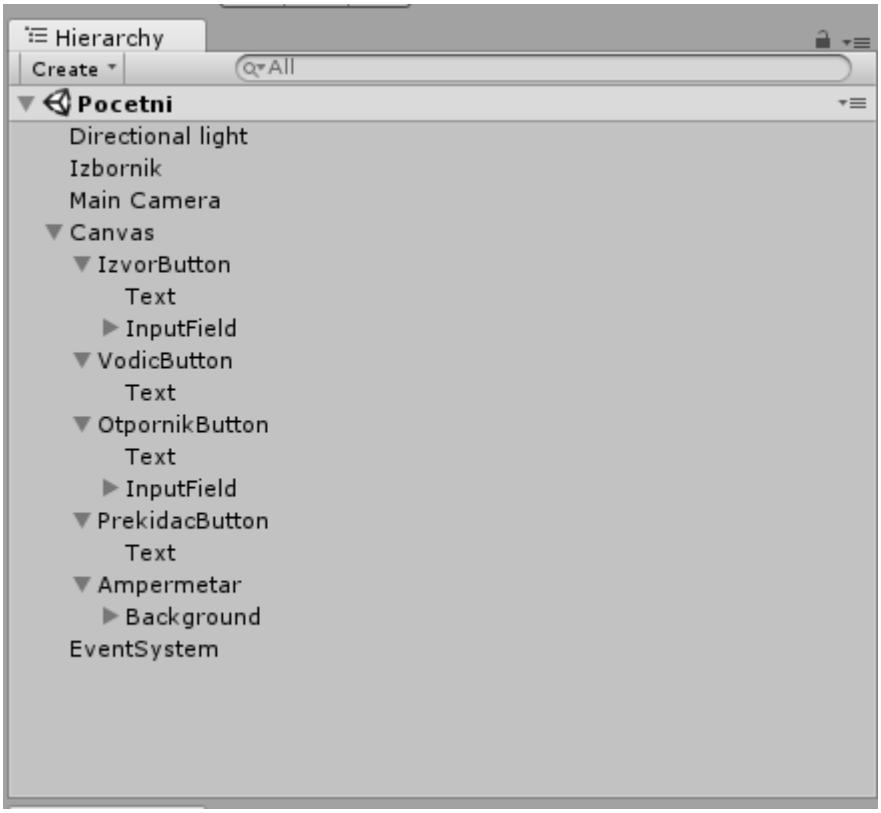
Klikom na *GameObject*, *UI* te *Button* stvara se gumb kao element potomak unutar *Canvas* objekta. Gumbovi, nakon što ih se označi u hijerarhiji, osim kartica *Transform* pod inspektor prozorom imaju kartice *Image (Script)* za odabir i upravljanje vizualnog dijela gumba i *Button (Script)* za upravljanje gumbom pomoću programskog koda. Potrebno je stvoriti četiri gumba i nazvati ih različitim imenima: *IzvorGumb*, *VodicGumb*, *OtpornikGumb* i *PrekidacGumb*. Pomoću ovih gumba sučelja korisnik će stvarati nove elemente strujnog kruga. Svaki objekt stvoren u svrhu sučelja izbornika potrebno je, odabirom u hijerarhiji, pomaknuti na željeno mjesto koristeći hvataljke u prozoru scene ili vrijednosti koordinatnih osi u pregledniku.



Slika 12. Pomicanje odabranog objekta u prozoru hijerarhije.

Zatim, klikom na *GameObject, UI* i *Inputfield* stvara se novi dio sučelja koji služi za unos teksta ili broja. *Inputfield* ili polje unosa ima osim vizualnih komponenti i potomke u obliku teksta i *placeholdera* koji je vidljiv u slučaju da tekst nije unesen. Koristan je potomak tekst koji je moguće iskoristiti u skripti i koristit će se kao način pomoću kojeg će korisnik unijeti željeni otpor otpornika i napon izvora. Stoga se stvaraju dva polja unosa te ih se povlači u prozoru hijerarhije na prikladne gumbe za otpornik i izvor. Povlačenjem objekta na objekt u prozoru hijerarhije stvara se veza roditelj-potomak te je time olakšan kasniji rad manipuliranjem objekata. Od dodanih elemenata sučelja nedostaje samo ampermetar, stvara ga se u obliku *UI* elementa *Toggle* odnosno prekidač. *UI* objekt *Toggle* unaprijed je predodređen s dva vizualna stanja koja je moguće mijenjati uvjetovano skriptom. Ideja je korisniku dati do znanja kad krugom poteče struja promjenom izgleda ampermetra. Pod prekidačem postoje potomci *Background* i *Checkmark*. Klikom na njih u inspektoru na raspolaganju stoji *kartica Image (Script)* pod kojima se može promijeniti *Source image* povlačenjem unaprijed pripremljenih slika[9] iz projekt prozora. Desnim klikom u projekt prozoru odabire se *Import asset* u izborniku i kreće se kroz mape do pripremljene slike koju se želi koristiti u programu. Sliku *Background* elementa potrebno je promijeniti u sliku A*mper_ugasen*, a sliku *Checkmark* elementa u *Amper_upaljen*. Svi stvoreni elementi postavljaju se u prostoru pomoću *Rect transform* kartice u *Inspectoru* nakon što se klikne na njih u hijerarhiji. Pozicija je proizvoljna uz preporuku lijeve strane, jednake veličine gumbova i prostora među njima te srednjeg poravnanja teksta gumbova. Dodatni vizualni dijelovi sučelja obrađeni su pri kraju izrade programa.

[9] Sličice su unaprijed izrađene od jednostavnih oblika u programu za obradu slika



Slika 13. Krajnji izgled prozora hijerarhije.

Skripta izbornika imala je funkcije koje se mogu izvršiti klikom na pojedini gumb izbornika. Klikom na bilo koji gumb u prozoru hijerarhije otvorit će se taj gumb u prozoru preglednika na čijem dnu je vidljiv dio menija koji započinje s *OnClick()*. Prvi izbornik određuje se na *Runtime Only*, te se iz prozora hijerarhije povuče izbornik na mjesto ispod izbornika. Time je učitana skripta s objekta izbornika i ponuđene su sve funkcije iz te skripte na korištenje gumbu. U desnom padajućem izborniku izabire se *Izbornik* i prikladna funkcija pri dnu liste, primjerice *StvoriIzvor* za gumb izvora. Isto je potrebno učiniti za sve ostale gumbe izbornika.

Potrebno je stvoriti novu C# skriptu nazvanu *ampermetar* te ju se povlači na objekt ampermetra u prozoru hijerarhije. Unutar skripte piše se sljedeći programski kod:

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class Ampermetar : MonoBehaviour {

	private float otpor = 0.0f;
	private float napon = 0.0f;
	private float struja = 0.0f;
	public Text gotoviAmp;
	private Rect windowRect = new Rect(0, 0, 150, 60);
	private Rect rRect = new Rect(5, 0, 140, 20);
	private Rect vRect = new Rect(5, 20, 140, 20);
	private Rect iRect = new Rect(5, 40, 140, 20);

	private bool prozorOcitanja = false;

	public void OnGUI() {
		if (prozorOcitanja) {
			GUI.Window(0, windowRect, IzmjeriOtpor, "", guiStyle);}
	}
	private void IzmjeriOtpor(int windowID) {
		guiStyle.fontSize = 20;
		GUI.Label(rRect, "Otpor: " + otpor.ToString() + " Ω", guiStyle);
		GUI.Label(vRect, "Napon: " + napon.ToString() + " V", guiStyle);
		GUI.Label(iRect, "Struja: " + struja.ToString() + " A", guiStyle);}

	public void PrikaziProzor(OtpornikSpoj r) {
		prozorOcitanja = true;

		Vector3 screenPos =Camera.main.WorldToScreenPoint(r.transform.position);
		windowRect.x = screenPos.x - 50;
		windowRect.y = Screen.height - (screenPos.y + 80.0f);
		windowRect.width = 150;
		windowRect.height = 60;

		otpor = Mathf.Round(r.otpor * 100.0f) / 100.0f;
		napon = Mathf.Round(r.PadNapona * 100.0f) / 100.0f;
		struja = Mathf.Round(r.struja * 100.0f) / 100.0f;}

	public void SakrijProzor() {
		prozorOcitanja = false;}
}
```

Skriptna datoteka 2. Ampermetar.

Objekt ampermetra služi kao vizualni pokazatelj prisutnosti toka struje u strujnom krugu te drži logiku za stvaranje prozora očitanja pada napona, otpora i struje kroz otpornik. Početni dio skripte sadrži varijable za pohranu otpora, napona i struje te dimenzije prozora kojima se prikazuju očitanja. Prozor očitanja stvara se funkcijom *OnGUI*(), kao meni u kojem se prikazuje dinamični tekst koji je uvijek prisutan na sceni, ali je skriven. Funkcijom *IzmjeriOtpor* uzima se vrijednosti otpora, napona i struje iz drugih skripti i pretvara u tekst pogodan za prikaz na zaslonu. Funkcija *PrikaziProzor* poziva se u drugim skriptama i uloga joj je objašnjena imenom, isto vrijedi za funkciju *SakrijProzor*. Među tim funkcijama nalazi se programski kod za položaj prozora relativan

s kamerom i programski kod za zaokruživanje vrijednosti mjerenja elemenata. U slučaju da su potrebna preciznija mjerenja ovdje se mijenjaju vrijednosti.

4.5. Elementi strujnog kruga

Elementi strujnog kruga obrađeni su u dva dijela, sami elementi od trodimenzionalnih oblika izrađeni su u jednom poglavlju imena predlošci, a skripte i logika samog strujnog kruga obrađena je po svakom elementu posebno kako bi se istaknula uloga svakog elementa i ukazalo na moguće preinake i poboljšanja. Određene skripte služe kao predložak čiji programski kod podređene skripte preuzimaju i nadograđuju, stoga neće biti vezane za objekt. Ostale skripte vežu se uz prikladne objekte, odnosno predloške objekata nakon što ih se izradi. Skripte se mogu organizirati u mape, radi preglednosti projekta, stvaranjem mapi u prozoru projekta pomoću izbornika nakon desnog klika miša.

4.5.1. Spoj

Svaki element strujnog kruga ima dvije točke spoja, ulazna i izlazna, može ih se zamisliti kao priključke za vodiče u stvarnom kompletu strujnih krugova. Spojeve u programu simulira se dvjema sferama udaljenima za određenu dužinu. Provjeravajući jesu li sfere u dodiru, odnosno nalaze li se u istom prostoru, ujedno se provjerava i jesu li elementi prikladnih spojeva spojeni. Ako su dvije sfere u prijeklopu ili na istim koordinatama smatra se da su ostvarile spoj. Izrada samog spoja u prostoru obrađena je u sljedećim odjeljcima. Stvara se nova skripta, koju je kasnije, nakon što se izradi objekt spoja, potrebno povući na spoj, potrebno ju je otvoriti i unijeti sljedeće:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class Spoj : MonoBehaviour {

	private Vector3 prostorZaslona;
	private Vector3 pomak;
	private Grana	spoji;
	private bool  povlacenje = false;
	private int	indeks;
	private List<Spoj> spojeviIstePozicije = new List<Spoj>();
	public Grana spoj {
		get { return spoji; }
		set { spoji = value; }}
	public int Indeks {
		get { return indeks; }}
	public bool spojiSpojeve() {
		return spojeviIstePozicije.Count > 0;}

	public List<Spoj> IstiSp {
		get { return spojeviIstePozicije; }}

	void Start() {
		indeks = StrujniKrug.Instance.DodajSpoj(this);}

	void OnMouseDown() {
		if (Input.GetMouseButton(0)) {
		prostorZaslona = Camera.main.WorldToScreenPoint(transform.position);
		pomak = transform.position - Camera.main.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y, prostorZaslona.z));
		povlacenje = true;
	StrujniKrug.Instance.nadjiSpojeveIstePozicije(this, spojeviIstePozicije);}}

	void OnMouseDrag() {
		if (povlacenje) {
		Vector3 curScreenSpace = new Vector3(Input.mousePosition.x,
Input.mousePosition.y, prostorZaslona.z);
		Vector3 curPosition = Camera.main.ScreenToWorldPoint(curScreenSpace) +
pomak;
		transform.position = StrujniKrug.Instance.StickySpojPos(this,
curPosition, spojeviIstePozicije);
			spoji.gameObject.SendMessage("JunkUpdate", this);}
	}

	void OnMouseUp() {
		spojeviIstePozicije.Clear();
		StrujniKrug.Instance.ponoviSpojeveIstePozicije();
		StrujniKrug.Instance.RK.Primijeni();
		povlacenje = false;}
}
```

Skriptna datoteka 3. Spoj.

Prva polovica skripte zaslužna je za logiku liste spojeva. Spojeve koji se čine u strujnom krugu zapisuju se u listu i pohranjuju kako bi se daljnjim skriptama mogla provjeriti struktura strujnog kruga. Osim što se spojevi popisuju dodjelom indeksnog broja, odmah je potrebno provjeriti nalaze li se sfere spojeva u istom prostoru te se poziva logika strujnog kruga obrađena dalje u radu. U drugoj polovici skripte definiraju se funkcije pomoću kojih se manipulira spojevima. Funkcija

OnMouseDown() ugrađena je u *Unity* i poziva se nakon što se mišem klikne na spoj, unutar funkcije zapisana je logika provjere nalazi li se povučena točka u istom prostoru kao i neka druga točka, odnosno jesmo li ju spojili. Funkcija *OnMouseDrag*() također je ugrađena funkcija koja se poziva, u ovom slučaju, na povlačenje miša. Unutar funkcije zapisana je logika pomicanja jedne točke spoja, dokle god je tipka miša pritisnuta, točka spoja mijenja svoje koordinate paralelno s pokazivačem miša. Nakon ostvarenog spoja šalju se upute drugim skriptama za ažuriranje liste spojeva. Zadnja funkcija *OnMouseUp*(), pozvana nakon što se pusti lijeva tipka miša, primjenjuje promjene strujnom krugu i čisti spremnik provjera spojeva istih pozicija.

4.5.2. Grana

Granu ili vod strujnog kruga čini svaki element kojim struja teče, logika kruga nalaže da svaki element strujnog kruga stoga ima zajedničku ulogu voda. Skriptom se problem rješava uvođenjem grane kao zasebne skripte koju će svi ostali elementi kruga naslijediti i koristiti kao temelj logike. Time je moguće ažurirati i provjeravati liste i matrice logike strujnog kruga bez obaziranja na vrstu elementa ili drugih karakteristika.

Unutar skripte najprije se postavljaju varijable kao što su pozicije spojeva obrađenih u prošlom odlomku. Jedna od varijabli je i početni otpor koji se postavlja u slučaju da nije definirano kasnije. Zaključak je da svaki dio strujnog kruga time ima otpor toliko malen da nestaje u zaokruživanju pri mjerenju, no ipak dovoljan da olakšava izračun struje pri mjerenju. Zatim se pozivaju razne funkcije koje su ranije definirane, kao što su mjerenje iz skripte ampermetra te ažuriranje spojeva iz skripte spojeva. Funkcija *Start()*, koja se poziva stvaranjem objekta, postavlja struju i pad napona na nulu. Prije postavljanja strujnog kruga stvaraju se sfere spojeva i naposljetku, stvara se *BoxCollider* čija uloga je služiti kao hvatište prilikom povlačenja miša. Povećanjem istog moguće je poboljšati osjećaj preciznosti pomicanja elemenata strujnog kruga pri korištenju programa, primjerice ukoliko se umjesto miša koristi dodirna površina *smartphone* uređaja ili tableta.

```
using UnityEngine;
using System.Collections;

public abstract class Grana : MonoBehaviour {

	protected Vector3 startLinkPos = new Vector3(-0.9f, 0.0f, 0.0f);
	protected Vector3 endLinkPos = new Vector3(0.9f, 0.0f, 0.0f);
	protected BoxCollider  boxCollider;
	private Vector3 screenSpace;
	private Vector3 offset;
	public Spoj startJunction;
	public Spoj endJunction;
	public float otpor = 0.000001f;

	protected abstract void RenderInit();
	protected abstract void Render(Vector3 start, Vector3 end);
	protected abstract void Mjeri(bool show);
	public abstract void JunkUpdate(Spoj jn);
	public float struja { get; set; }
	public float PadNapona     { get; set; }

	public void RecalcJunctionPostion() {
		Vector3 s = transform.TransformPoint(startLinkPos);
		Vector3 e =  transform.TransformPoint(endLinkPos);
		Render(s, e);
		startJunction.transform.position = s;
		endJunction.transform.position = e;}

	void Start () {
		transform.position = Vector3.zero;
		boxCollider = (BoxCollider) gameObject.AddComponent<BoxCollider>();
		if(boxCollider== null) {
			Debug.Log("null lineRenderer");}
		boxCollider.center = Vector3.zero;
		boxCollider.size = new Vector3(1.6f, 0.6f, 0.1f);
		RenderInit();
		startJunction       =       Instantiate(startJunction,       startLinkPos,
Quaternion.identity) as Spoj;
		endJunction = Instantiate(endJunction, endLinkPos, Quaternion.identity) as
Spoj;
		startJunction.spoj = this;
		endJunction.spoj = this;
		struja = 0.0f;
		PadNapona = 0.0f;
		StrujniKrug.Instance.DodajGranu(this);}
```

Skriptna datoteka 4. Grana 1.dio.

Veoma važan dio funkcije *Start*() je zadnji red programskog koda u skriptnoj datoteci 4. Svako stvaranje grane poziva se na funkciju *DodajGranu*() koja se nalazi u skripti *StrujniKrug* obrađena dalje u radu. Tom funkcijom ažurira se popis svih elemenata strujnog kruga koji se kasnije koriste pri izradi računa za strujni krug.

```
	void OnMouseDown() {
		if (Input.GetMouseButton(0)) {
			screenSpace = Camera.main.WorldToScreenPoint(transform.position);
			offset = transform.position - Camera.main.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y, screenSpace.z));}
	}

	void OnMouseDrag() {
		Vector3 curScreenSpace = new Vector3(Input.mousePosition.x,
Input.mousePosition.y, screenSpace.z);
		Vector3 curPosition = Camera.main.ScreenToWorldPoint(curScreenSpace) +
offset;
		transform.position = curPosition;
		RecalcJunctionPostion();}

	void OnMouseOver() {
		if (Input.GetMouseButtonDown(1)) {
			Mjeri(true);}
		if (Input.GetMouseButtonUp(1)) {
			Mjeri(false);}
    }
}
```

Skriptna datoteka 5. Grana 2.dio.

4.5.3. Element kruga

Skripta elementa kruga sadrži zajedničke karakteristike svih elemenata strujnog kruga koje nisu nužno obuhvaćene prethodnom skriptom grane. Skripta ujedno nasljeđuje programski kod iz skripte grane vidljivo u prvoj liniji programskog koda poveznicom dvotočke u sintaksi. Programski kod elementa kruga određuje udaljenost spojnih točaka koje se koriste za spajanje elemenata te nasljeđivanje koordinata spojeva u slučaju uspješnog spajanja. U slučaju da je potrebno promijeniti relativne pozicije spojnih sferi svih elemenata kruga isto činimo promjenom varijable *duljina*.

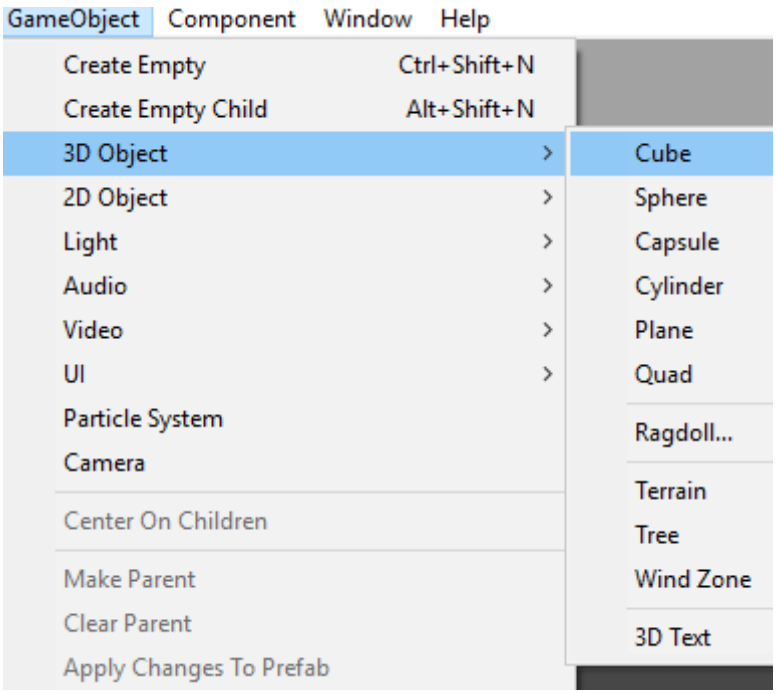
```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class ElementKruga : Grana {

    protected float duljina = 1.0f;

       protected override void RenderInit() {}
       protected override void Render(Vector3 start, Vector3 end) {}
       protected override void Mjeri(bool prikazi) {}

       public override void JunkUpdate(Spoj jn) {

              Vector3 dir = new Vector3(0.0f, 0.0f, 0.0f);
              if (startJunction.Equals(jn)) {
                     dir = startJunction.transform.position - endJunction.transform.position;

                     startJunction.transform.position = endJunction.transform.position +
duljina * dir.normalized;
              } else if (endJunction.Equals(jn)) {
                     dir = endJunction.transform.position - startJunction.transform.position;

                     endJunction.transform.position = startJunction.transform.position +
duljina * dir.normalized;
              } else {
                     return;}

              Vector3 rotDir = endJunction.transform.position -
startJunction.transform.position;
              Quaternion q = Quaternion.FromToRotation(new Vector3(1.0f, 0.0f, 0.0f),
rotDir.normalized);
              boxCollider.transform.rotation = q;
              boxCollider.transform.position = (startJunction.transform.position +
endJunction.transform.position) * 0.5f;
       }
}
```

Skriptna datoteka 6. Element kruga.

4.5.4. Predlošci

Predlošci ili *Prefabs* ugrađena je mogućnost programa Unity pomoću koje se objekti mogu postaviti u konfiguraciji te sačuvati unutar programa za učestalo korištenje pomoću skripti. Ukoliko je unaprijed poznato koji objekti se žele stvarati i uništavati potrebno ih je sačuvati u obliku predloška, na taj način se olakšava praćenje rada i smanjuje opterećenost spremnika programa. Elemente strujnog kruga potrebno je stvarati po potrebi stoga se za svaki element radi predložak. Koristeći karticu *GameObject*, *3DObject* na alatnoj traci pruža mogućnost stvaranja pravilnih geometrijskih tijela. Promjenom koordinata i dimenzija istih može se ugrubo načiniti shematski prikaz svakog elementa strujnog kruga. Potrebno je stvoriti predloške za izvor, otpornik, sklopku, vodič i same spojke. Objekti stvoreni na ovaj način nastaju u prozoru hijerarhije gdje ih je moguće dodatno manipulirati. Najprije se stvara *EmpyObject*, kojeg je potrebno preimenovati u *IzvorSpoj*, zatim

OtpornikSpoj, *VodicSpoj* i naposljetku *SklopkaSpoj*, koji se koriste kao držači geometrijskih likova i unaprijed određenih skripti. Koordinate i dimenzije ovih praznih objekata nisu bitne. Zatim je potrebno stvoriti kocke i kvadre koji se rotiraju i postupno pomiču s ciljem stvaranja shematskog prikaza određenog elementa. Važno je imati na umu udaljenost između spojeva postavljenih u prethodnim skriptama kako bi ostao poznat prostor kojim se raspolaže.

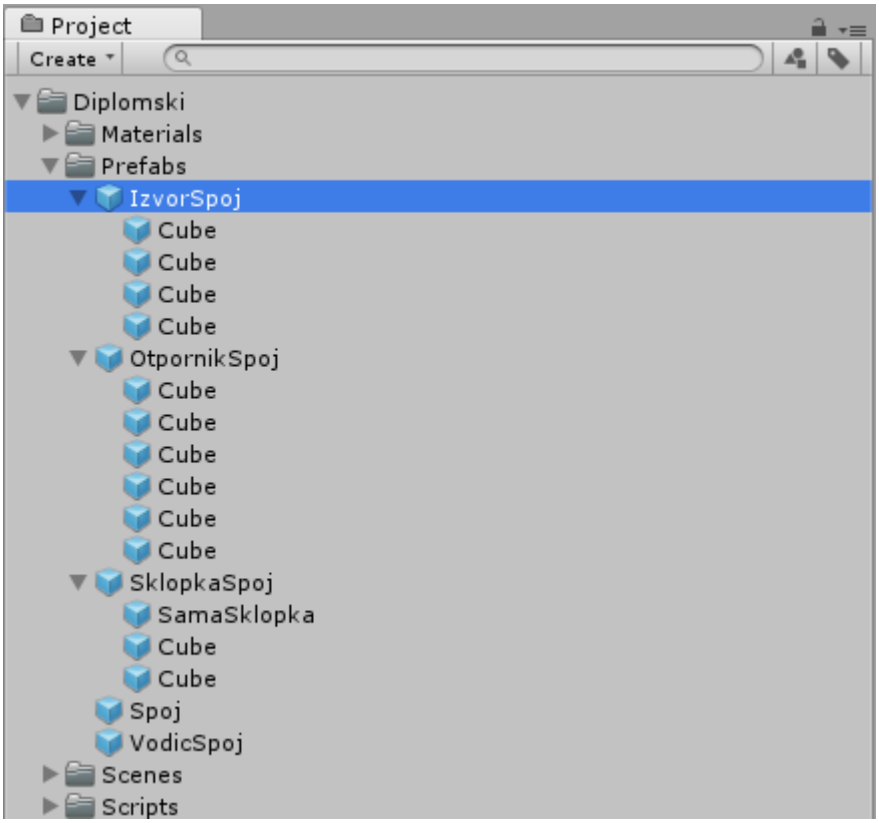


Slika 14. Stvaranje ugrađenih 3D objekata.

Nakon što su elementi oformljeni ostavljaju se u prozoru hijerarhije dok prikladne skripte nisu napisane i podređene. Element *Spoja* zaseban je i u ovom slučaju ima izgled sfere koja se stvara pomoću alatne trake ponovnim odabirom *GameObject*, *3DObject*, *Sphere* te preimenovanjem u *Spoj*. Potrebno ju je označiti i pogledom u prozor preglednika vidljivi se izbornici za polumjer sfere te standardne kartice s dimenzijama i koordinatama. Koordinate nisu važne jer se ponovno radi predložak koji se postavlja na scenu skriptom, ali dimenzije sfere mijenjaju se pomoću *Scale* retka postavljajući ih na 0.2 u svakoj osi.

Sphere Collider kartica određuje dimenzije nevidljivog elementa sfere koji određuje koliko je veliko područje za hvatanje spoja. Ukoliko se želi povećati preciznost hvatanja spojeva isto je ovdje moguće učiniti. Vodič je prazan novostvoren objekt koji služi kao nosač skripte. Nakon što su napisane skripte za pojedine elemente, povlači ih se na elemente strujnog kruga u hijerarhiji te se sami elementi povlače u prozor projekta u mapu *Prefabs*. Time elementi postaju predlošci, u prozoru hijerarhije njihov tekst postaje plave boje što daje na znanje da je element sada instanca ili

inačica predloška. Bilo koja promjena predloška naslijeđena je od strane inačice te omogućava brze promjene bez individualnih ugađanja.



Slika 15. Krajnji izgled mape predlošci.

Nakon izrade svih predložaka potrebno je vratiti se na objekt izbornika i popunjavati odabrana mjesta u skripti prozora preglednika s izrađenim predlošcima čime se ukazuje na objekte čije inačice se mogu stvoriti klikom na gumbe izbornika.

U nastavku je prikazan i pojašnjen programski kod svakog pojedinog elementa strujnog kruga. Nakon što je skripta napisana potrebno ju je povući na pripadnu predlošku elementa. Klikom na predložak u prozoru preglednika vidljiva je kartica skripte i sve varijable, predmeti i objekti koje su pomoću rječice *Public* u programskom kodu učinjene dostupnom cijelom programu. Svaka skripta traži da joj se definira početni i krajnji spoj povlačenjem predloška spoja.

4.5.5. Vodič

Element Vodiča razlikuje se od ostalih elemenata jer se od njega zahtijeva mogućnost promjene dimenzija i oblika kako bi bilo moguće spajati ostale elemente. Nasljedstvo skripti stoga ne ide pomoću skripte *ElementKruga* već izravno iz skripte *Grana*. Sam element vodiča sastoji se od dvije sfere spoja i obojanog pravca koji ih spaja u svakom trenutku. Otpor vodiča naslijeđen je od grane te ukoliko ga se želi promijeniti potrebno je dodati redak programskog koda u sljedeću skriptu.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;

class VodicSpoj : Grana {

	private LineRenderer lineRenderer;
	private Vector3 pocetakPravca = new Vector3(-0.9f, 0.0f, 0.0f);
	private Vector3 krajPravca = new Vector3(0.9f, 0.0f, 0.0f);
	public Material mat = null;

	private class RKrugaInput : RKrugPrijemnik {
		VodicSpoj vodic;
	 public RKrugaInput(VodicSpoj v) {
			vodic = v;}
		 public void RKrugaGotovo() {
		GameObject.Find("Ampermetar").GetComponent<Toggle>().isOn = false;

		if (Mathf.Abs(vodic.struja) > 0.05f) {
		    GameObject.Find("Ampermetar").GetComponent<Toggle>().isOn = true;}
		 }}

	private RKrugaInput rjKrugaInput;

	protected override void RenderInit() {
	 lineRenderer = (LineRenderer)gameObject.AddComponent<LineRenderer>();
	 lineRenderer.startWidth = 0.2f;
	 lineRenderer.endWidth = 0.2f;
	 lineRenderer.startColor = new Color(0, 0, 0, 1);
	 lineRenderer.endColor = new Color(0, 0, 0, 1);
	 lineRenderer.material = mat;

		lineRenderer.SetPosition(0, pocetakPravca);
		lineRenderer.SetPosition(1, krajPravca);

		rjKrugaInput = new RKrugaInput(this);
		StrujniKrug.Instance.RK.DodajPrijemnik(rjKrugaInput);
	}
```

Skriptna datoteka 7. VodicSpoj 1.dio.

Početak skripte zauzet je varijablama koje sadrže početak i kraj obojanog pravca te materijal, odnosno sjenčanje istog. Funkcija *RKrugaInput*() komunicira s logikom strujnog kruga te daje krugu na znanje svaki spoj ostvaren vodičima. Funkcija *RKrugaGotovo*() poziva se u slučaju

pronalaska rješenja kruga odnosno uspostave strujnog kruga te šalje poruku paljenja odnosno gašenja elementa ampermetra koji su uspostavljeni ranije u radu. Funkcija imena *RenderInit*() ugrađena je funkcija koja se poziva pri prvom iscrtavanju slike programa i postavlja duljinu, boju i koordinate početka i kraja pravca koji predstavlja vodič.

```
	protected override void Render(Vector3 pocetak, Vector3 kraj) {
		lineRenderer.SetPosition(0, pocetak);
		lineRenderer.SetPosition(1, kraj);
	}

	protected override void Mjeri(bool prikazi) {}

	public override void JunkUpdate(Spoj jn) {

	Render(startJunction.transform.position, endJunction.transform.position);
	Vector3 dir = endJunction.transform.position - startJunction.transform.position;

	Quaternion q = Quaternion.FromToRotation(new Vector3(1.0f, 0.0f, 0.0f),
dir.normalized);
		boxCollider.transform.rotation = q;
		boxCollider.transform.position = (startJunction.transform.position +
endJunction.transform.position) * 0.5f;

	Vector3 newSize = boxCollider.transform.InverseTransformDirection(dir);
	boxCollider.size = new Vector3(newSize.x, 0.3f, 0.1f);

	startLinkPos =
boxCollider.transform.InverseTransformPoint(startJunction.transform.position);
	endLinkPos =
boxCollider.transform.InverseTransformPoint(endJunction.transform.position);}
}
```

Skriptna datoteka 8. VodicSpoj 2.dio.

U nastavku skripte, funkcijom *JunkUpdate* na naizgled složen način komunicira se sa skriptama u drugim dijelovima programa kako bi sustav provjere i rješavanja strujnog kruga obavijestili o promjeni spojeva kad god su prepoznata dva spoja na istim koordinatama.

4.5.6. Izvor

Element Izvora također nasljeđuje značajke skripte *ElementKruga*. Izuzev naslijeđenih varijabli potrebno je postaviti napon izvora, te odrediti početne koordinate i udaljenost spojeva ovog specifičnog elementa. Daljnje naredbe vezane za izvor strujnog kruga preuzimaju druge skripte programa. Ukoliko dosad nije bio izrađen sada se izrađuje element izvora pomoću četiri kvadra prilagođenih dimenzija te se sve povuče pod prazni objekt roditelj nazvan *IzvorSpoj*. Skripta se

povlači na objekt *IzvorSpoj* u prozoru hijerarhije te se cijeli roditelj objekt pretvara u predložak povlačenjem u mapu *Prefabs* u prozoru projekta.

```
using UnityEngine;
using System.Collections;

public class IzvorSpoj : ElementKruga {

	public float napon = 1.5f;

	protected override void RenderInit() {
		startLinkPos = new Vector3(-0.9f, 0.0f, 0.0f);
		endLinkPos = new Vector3(0.9f, 0.0f, 0.0f);
		duljina = 1.8f;}
}
```

Skriptna datoteka 9. IzvorSpoj.

4.5.7. Otpornik

Element Otpornika također se oblikuje pomoću gotovih objekata alatne trake. Klikom na *GameObject*, *3DObject*, *Cube* stvara se kocka te klikom i povlačenjem na *Scale* određenih osi u prozoru preglednika pod karticom *Transform* uređuje se izgled kocke. Izgled se manipulira, kao i programski kod prethodnih elemenata dok se ne postigne željeni izgled. Skripta ima definiranu funkciju *Mjeri*() pomoću koje se javlja skripti objekta ampermetra da stvori prozor s mjerenjima otpornika: naponom, strujom i otporom. Ostala obilježja element otpornika nasljeđuje od skripti element kruga i grana.

```
using UnityEngine;
using System.Collections;

public class OtpornikSpoj : ElementKruga {

	protected override void Mjeri(bool prikazi) {
		GameObject obj = GameObject.Find("Ampermetar");
		if (obj) {
			if (prikazi)
				obj.SendMessage("CreatePopUp", this);
			else
				obj.SendMessage("ClosePopUp", this);}
	}
}
```

Skriptna datoteka 10. OtpornikSpoj.

4.5.8. Sklopka

Pri izradi trodimenzionalnog objekta Sklopke, prate se metode prethodnih elemenata. Razlika se radi u kvadru koji igra ulogu same sklopke. Potrebno je izraditi dvije skripte koje se vežu za različite dijelove sklopke. Skripta Sklopka sadrži programski kod za funkciju *Ugasen*() koja se poziva klikom na sklopku pri korištenju programa. Funkcija traži objekt dijete koji je nazvan *SamaSklopka*, što je veoma važno replicirati, te mijenja kut objekta, odnosno fizički ga zakrene. Skriptu se povlači na prazni objekt nazvan *SklopkaSpoj* koji sadrži ostale kvadre ostatka sklopke. Prije pretvaranja objekta u predložak izrađuje se još jedna skripta.

```
using UnityEngine;
using System.Collections;

public class SklopkaSpoj : ElementKruga {
	public bool Ugasen() {
		Transform sklopTijelo = transform.FindChild("SamaSklopka");
		float kut = Vector3.Angle(transform.right, sklopTijelo.right);
		return (kut > -3.0f) && (kut < 10.0f);}
}
```

Skriptna datoteka 11. SklopkaSpoj.

Skripta u nastavku postavljena je na dijete element sklopke, dio koji se pomiče klikom, nazvan *SamaSklopka*. Unutar skripte postavljene su najprije varijable u kojima je važno pamtiti izvorne koordinate, stanje i pomak sklopke. Funkcija *sklopka()* upravlja pomicanjem pomičnog dijela sklopke i poziva ju se pomoću ugrađene funkcije *OnMouseDown*() u nastavku klikom miša. Ostale funkcije predstavljaju mjere opreza provjeravajući riješenost strujnog kruga na klik miša, puštanje lijeve tipke miša pa čak i prelaska miša preko elementa.

Izradom predloška posljednjeg elementa strujnog kruga potrebno je vratiti se na izbornik u prozoru hijerarhije. On se označi lijevim klikom te je potrebno pronaći dodanu skriptu u prozoru preglednika. Retke za predloške elemenata potrebno je popuniti predlošcima iz prozora projekta te se na taj način postavljaju odabrani predlošci kao objekti koji se stvaraju klikom na tipke korisničkog sučelja.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class Sklopka : MonoBehaviour {

	private Vector3 prostorZaslona;
	private Vector3 pomak;
	private bool upaljen = false;
	private Quaternion izvorniKvat;
	private Vector3 izvorniPolozaj;

	public void sklopka() {
		upaljen = !upaljen;
		if (upaljen) {
			izvorniPolozaj = transform.localPosition;
			izvorniKvat = transform.rotation;

			transform.rotation = transform.parent.rotation;
			transform.localPosition = new Vector3(0.0f, 0.0f, 0.0f);}
		else {
			transform.localPosition = izvorniPolozaj;
			transform.rotation = izvorniKvat;}
	}
	void Start() {
		izvorniPolozaj = transform.localPosition;
		izvorniKvat = transform.rotation;}
	void OnMouseDown() {
		sklopka();}
	void OnMouseOver() {
		if (Input.GetMouseButtonUp(0)) {
			StrujniKrug.Instance.RK.Primijeni();}
	}
	void OnMouseUp() {
		StrujniKrug.Instance.RK.Primijeni();}
	void OnMouseExit() {
		StrujniKrug.Instance.RK.Primijeni();}
}
```

Skriptna datoteka 12. Sklopka.

4.6. Algoritam za rješavanje logike strujnog kruga

Algoritam rada strujnog kruga podijeljen je u tri dijela. Prilikom izrade elemenata strujnog kruga potrebno je pratiti svaki element i njegove spojeve, sfere na stranama elemenata pomoću kojih se elementi međusobno spajaju, te ih unositi u spremnike u obliku jedne ili više lista. Osim spremnika elemenata i njihovih dijelova, najveće poteškoće digitalizacije strujnog kruga nastaju zbog podjele vodova strujnog kruga na grane, mijenjajući time vrijednosti struje i napona ovisno o elementima u granama. Potrebno je stoga stvoriti sustav logike koji će pratiti grane strujnog kruga i postavljati matematička ograničenja pri računu struje i pada napona koristeći otpore svakog elementa i napona izvora. Račun je odvojen u posljednji dio sustava skripti za rješavanje kruga. Proces teče sljedećim redom: prilikom stvaranja, pomicanja i povezivanja elemenata sustav strujnog kruga prati i pamti sve elemente, spajanjem sustav matrica stvara dvodimenzionalnu sliku grana i vodova strujnog

kruga i naposljetku komunicirajući s prva dva sustava, sustav rješenja određuje glavne elemente kruga i računa potrebne vrijednosti za rješenje.

4.6.1. Strujni krug

Sustav prati popise svih spojeva i grana strujnog kruga prilikom korištenja softvera. Osim varijabli sadrži funkcije za osvježavanje svih popisa, kao što su: *DodajSpoj*() i *DodajGranu*().

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class StrujniKrug : MonoBehaviour {

	List<Spoj>    popisSpojeva;
	List<Grana>   popisGrana;

	SustavRjesenja      rjes;

	public List<Spoj> LS {
		get { return popisSpojeva; }
	}
	public List<Grana> LG {
		get { return popisGrana; }
	}
	public RijesiKrug RK {
		get { return rjes; }
	}
	static StrujniKrug _instance;
	void Awake () {
		popisSpojeva  = new List<Spoj>();
		popisGrana    = new List<Grana>();
		rjes          = new SustavRjesenja();}
	public int DodajSpoj (Spoj sp) {
		popisSpojeva.Add(sp);
		return popisSpojeva.IndexOf(sp);}

	public int DodajGranu (Grana gr) {
		popisGrana.Add(gr);
		return popisGrana.IndexOf(gr);}
```

Skriptna datoteka 13. StrujniKrug 1. dio.

U nastavku skripte definiraju se funkcije pomoću kojih se vrši provjera dodira dva spoja različitih elemenata svakog prijelaza mišem preko vodiča ili prekidača. Prilikom provjere uspoređuju se koordinate svakog spoja i, ovisno o rezultatima, premještaju se između liste neovisnih spojeva i liste spojenih spojeva vidljivo u funkcijama *StickySpojPos*() i *nadjiSpojeveIstePozicije*(). Kraj skripte stvara instancu strujnog kruga kao privremeni objekt kako bi se omogućila komunikacija s ostalim skriptama.

```
public Vector3 StickySpojPos (Spoj sp, Vector3 pos, List<Spoj> spList) {
	List<Spoj> neovisniSpojevi = new List<Spoj>();
	foreach (Spoj spi in popisSpojeva) {
		if (sp.Equals(spi) || sp.spoj.Equals(spi.spoj))
			continue;
		bool ukloniListu = false;
		foreach (Spoj sj in spList) {
			if (sj.Equals(spi)) {
				ukloniListu = true;
				break;}
			}
			if (ukloniListu)
				continue;
			neovisniSpojevi.Add(spi);}
		float maxSqDist = 1.0f;
		foreach (Spoj spi in neovisniSpojevi) {
			float distSQ = (pos - spi.transform.position).sqrMagnitude;
			if (distSQ < maxSqDist) {
				maxSqDist = distSQ;
				pos = spi.transform.position;}
		}
		return pos;}

public void nadjiSpojeveIstePozicije (Spoj sp, List<Spoj> spList) {
	foreach (Spoj spi in popisSpojeva) {
	if (sp.Equals(spi) || sp.spoj.Equals(spi.spoj))
				continue;
	float distSQ = (sp.transform.position - spi.transform.position).sqrMagnitude;
		if (distSQ < 0.01f) {
			if (!spList.Contains(spi)) {
				spList.Add(spi);}
			}	}	}

public void ponoviSpojeveIstePozicije () {
	foreach (Spoj jni in popisSpojeva) {
		jni.IstiSp.Clear();
		nadjiSpojeveIstePozicije(jni, jni.IstiSp);}
	}
public static StrujniKrug Instance {
	get {if (_instance == null) {
		_instance = FindObjectOfType(typeof(StrujniKrug)) as StrujniKrug;
			if (_instance == null) {
			_instance = new GameObject("Circuit System",
typeof(StrujniKrug)).GetComponent<StrujniKrug>();}
			}
			return _instance;}
	}	}
```

Skriptna datoteka 14. StrujniKrug 2. dio.

4.6.2. Matrice

Kako bi riješili problem grananja potrebno je vodove prikazati dvodimenzionalno pomoću matrica te samu logiku odraditi pomoću matričnog računa.

```
using UnityEngine;
using System.Collections;

public class Matrica2D {
	public float[,] podaciMatrice = new float[2, 2];
	Matrica2D() {
		podaciMatrice[0, 0] = podaciMatrice[0, 1] = podaciMatrice[1, 0] =
podaciMatrice[1, 1] = 0.0f;
	}
	Matrica2D(float m00, float m01, float m10, float m11) {
		podaciMatrice[0, 0] = m00;
		podaciMatrice[0, 1] = m01;
		podaciMatrice[1, 0] = m10;
		podaciMatrice[1, 1] = m11;}

	public static Matrica2D primiRotaciju(float radijani)
	{
		float cosKut = Mathf.Cos(radijani);
		float sinKut = Mathf.Sin(radijani);
		return new Matrica2D(cosKut, -sinKut, sinKut, cosKut);}

	public static Vector2 operator*(Matrica2D A, Vector2 B)
	{
		return new Vector2(A.podaciMatrice[0, 0]*B.x + A.podaciMatrice[0, 1]*B.y,
A.podaciMatrice[1, 0]*B.x + A.podaciMatrice[1, 1]*B.y);}	}
```

Skriptna datoteka 15. Matrica2D.

Prva skripta matrica pomoćna je skripta koja se poziva u radu glavne skripte prikazane u nastavku. Najjednostavnije objašnjenje rada skripte je pretvorba podataka iz oblika varijable programskog jezika C# u varijable funkcije koja će nadalje igrati ulogu matrice.

Skripta u nastavku započinje definicijom varijabli redaka, stupaca, matrica koji se koriste kao spremnici za obradu. Prije definicija funkcija pomoću kojih se matrice pretvara u druge oblike ili mijenja tijekom rada, potrebno je definirati posebne slučajeve kao što su nul i kvadratna matrica.

```
using UnityEngine;
using System.Collections;

public class Matrica {
    public int redak;
    public int stupac;
    public float[,] mat;

    public Matrica L;
    public Matrica U;
    private int[] pi;
    private double determinantaP = 1;

    public Matrica(int iRedci, int iStupci)
    {
        redak = iRedci;
        stupac = iStupci;
        mat = new float[redak, stupac];}

    public bool Kvadratna()
    {
        return (redak == stupac);}

       public bool JeNula()
       {
              bool jeNula = true;
        for (int i = 0; i < redak; i++) {
            for (int j = 0; j < stupac; j++) {
                        if (Mathf.Abs(mat[i ,j]) > 0.01f) {
                               jeNula = false;
                               break;}
                  }
                  if (!jeNula) break;}
             return jeNula;}

    public float this[int iRedak, int iStupac]
    {get { return mat[iRedak, iStupac]; }
        set { mat[iRedak, iStupac] = value; }}

    public Matrica GetCol(int k)
    {Matrica m = new Matrica(redak, 1);
        for (int i = 0; i < redak; i++) m[i, 0] = mat[i, k];
        return m;}

    public void SetCol(Matrica v, int k)
    {for (int i = 0; i < redak; i++) mat[i, k] = v[i, 0];}
```

Skriptna datoteka 16. Matrica 1. dio.

Funkcijom *MakeLU*() matrica se pohranjuje za uporabu daljnjim funkcijama, stvaraju se i pohranjuju determinante matrice te ukratko provjeravaju članovi matrice za jednakosti i nul vrijednosti. U nastavku skripte definirane su razne funkcije za obradu matrica: inverz matrice, određivanje determinante, dupliciranje, zamjena elemenata, pretvaranje elemenata u nule, množenje, zbrajanje matrica itd. Neke od tih funkcija se neće uopće ni pozivati pri normalnom radu softvera, ali su potrebne za rijetke, granične slučajeve.

```
    public bool MakeLU()
    {
        L = IdentityMatrix(redak, stupac);
        U = Duplicate();
        pi = new int[redak];
        for (int i = 0; i < redak; i++) pi[i] = i;
        double p = 0;
        float pom2;
        int k0 = 0;
        int pom1 = 0;

        for (int k = 0; k < stupac - 1; k++)
        {
            p = 0;
            for (int i = k; i < redak; i++)
            {
                if (Mathf.Abs(U[i, k]) > p)
                {p = Mathf.Abs(U[i, k]);
                    k0 = i;}
            }
            if (p == 0) return false;

            pom1 = pi[k]; pi[k] = pi[k0]; pi[k0] = pom1;

            for (int i = 0; i < k; i++)
            {
                pom2 = L[k, i]; L[k, i] = L[k0, i]; L[k0, i] = pom2;
            }

            if (k != k0) determinantaP *= -1;

            for (int i = 0; i < stupac; i++){
                pom2 = U[k, i]; U[k, i] = U[k0, i]; U[k0, i] = pom2;}

            for (int i = k + 1; i < redak; i++)
            {
                L[i, k] = U[i, k] / U[k, k];
                for (int j = k; j < stupac; j++)
                    U[i, j] = U[i, j] - L[i, k] * U[k, j];}
        }

             return true;}


    public Matrica SolveWith(Matrica v)
    {  bool singular = false;
        if (L == null) {
                    singular = !MakeLU();}
             if (singular)
                    return ZeroMatrix(1, 1);

        Matrica b = new Matrica(redak, 1);
        for (int i = 0; i < redak; i++) b[i, 0] = v[pi[i], 0];
        Matrica z = SubsForth(L, b);
        Matrica x = SubsBack(U, z);
        return x;}
```

Skriptna datoteka 17. Matrica 2. dio.

```
public Matrica Invert()
    {if (L == null) MakeLU();

        Matrica inv = new Matrica(redak, stupac);

        for (int i = 0; i < redak; i++)
        {Matrica Ei = Matrica.ZeroMatrix(redak, 1);
            Ei[i, 0] = 1;
            Matrica col = SolveWith(Ei);
            inv.SetCol(col, i);
                    }
        return inv;
             }

    public double Det(){
        if (L == null) MakeLU();
        double det = determinantaP;
        for (int i = 0; i < redak; i++) det *= U[i, i];
        return det;
             }

    public Matrica GetP(){
        if (L == null) MakeLU();
        Matrica matrix = ZeroMatrix(redak, stupac);
        for (int i = 0; i < redak; i++) matrix[pi[i], i] = 1;
        return matrix;
             }

    public Matrica Duplicate()
    {Matrica matrix = new Matrica(redak, stupac);
        for (int i = 0; i < redak; i++)
            for (int j = 0; j < stupac; j++)
                matrix[i, j] = mat[i, j];
        return matrix;
             }

       public Matrica Minor(int row, int col)
       {Matrica res = new Matrica(redak - 1, stupac - 1);

       for (int r = 0; r < redak - ((row == redak) ? 1 : 0); ++r) {
              for (int c = 0; c < stupac - ((col == stupac) ? 1 : 0); ++c) {
              res[r - ((r > row) ? 1 : 0), c - ((c > col) ? 1 : 0)] = mat[r, c];}
              }
              return res;
              }

    public static Matrica SubsForth(Matrica A, Matrica b)
    {  if (A.L == null) A.MakeLU();
        int n = A.redak;
        Matrica x = new Matrica(n, 1);

        for (int i = 0; i < n; i++)
        {     x[i, 0] = b[i, 0];
            for (int j = 0; j < i; j++) x[i, 0] -= A[i, j] * x[j, 0];
            x[i, 0] = x[i, 0] / A[i, i];}
        return x;}
```

Skriptna datoteka 18. Matrica 3. dio.

```
    public static Matrica SubsBack(Matrica A, Matrica b)
        if (A.L == null) A.MakeLU();
        int n = A.redak;
        Matrica x = new Matrica(n, 1);

        for (int i = n - 1; i > -1; i--)
        {x[i, 0] = b[i, 0];
            for (int j = n - 1; j > i; j--) x[i, 0] -= A[i, j] * x[j, 0];
            x[i, 0] = x[i, 0] / A[i, i];}
        return x;}

    public static Matrica ZeroMatrix(int iRows, int iCols)
    {Matrica matrix = new Matrica(iRows, iCols);
        for (int i = 0; i < iRows; i++)
            for (int j = 0; j < iCols; j++)
                matrix[i, j] = 0;
        return matrix;}

    public static Matrica IdentityMatrix(int iRows, int iCols)
    {Matrica matrix = ZeroMatrix(iRows, iCols);
        for (int i = 0; i < Mathf.Min(iRows, iCols); i++)
            matrix[i, i] = 1;
        return matrix;}

    public static Matrica Transpose(Matrica m)
    {Matrica t = new Matrica(m.stupac, m.redak);
        for (int i = 0; i < m.redak; i++)
            for (int j = 0; j < m.stupac; j++)
                t[j, i] = m[i, j];
        return t;}

       public static float SimpleDeterminant(Matrica matrix)
    {if (matrix.redak != matrix.stupac)
                    return 0.0f;

              float  d = 0.0f;
              int    n = matrix.redak;
              if (n == 1) {
                    d = matrix[0, 0];}
else if (n == 2) {d = matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0];}
else {for (int c = 0; c < matrix.stupac; ++c) {Matrica M = matrix.Minor(0, c);
              d += ((c + 1)%2 + (c + 1)%2 - 1) * matrix[0, c] * SimpleDeterminant(M);}
              }
        return d;}

    private static Matrica Multiply(float n, Matrica m)
    {Matrica r = new Matrica(m.redak, m.stupac);
        for (int i = 0; i < m.redak; i++)
            for (int j = 0; j < m.stupac; j++)
                r[i, j] = m[i, j] * n;
        return r;}
    private static Matrica Add(Matrica m1, Matrica m2)  {
        Matrica r = new Matrica(m1.redak, m1.stupac);
        for (int i = 0; i < r.redak; i++)
            for (int j = 0; j < r.stupac; j++)
                r[i, j] = m1[i, j] + m2[i, j];
        return r;}}
```

Skriptna datoteka 19. Matrica 4. dio.

4.6.3. Rješavanje kruga

Prva skripta rješenja kruga, nazvana *RijesiKrug*, sadrži dijelove programskog koda koji su odgovorni za upravljanje svim elementima kruga, izračun veličina unutar strujnog kruga te primjenu izračunatog rješenja na zadane elemente kruga.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public interface RKrugPrijemnik {
	void RKrugaGotovo();}

public abstract class RijesiKrug {
	private List<RKrugPrijemnik> prijemnici = new List<RKrugPrijemnik>();
	public abstract void Primijeni();
	public void DodajPrijemnik(RKrugPrijemnik csl) {
		prijemnici.Add(csl);}
	public void UkloniPrijemnik(RKrugPrijemnik csl) {
		prijemnici.Remove(csl);}
	protected void FKrugRijesen() {
		foreach(RKrugPrijemnik prijemnik in prijemnici) {
			prijemnik.RKrugaGotovo();}}
}

public class SustavRjesenja : RijesiKrug {

	public class IzvorAdap : LinearnoRKruga.Izvor {
		IzvorSpoj izvor;
		public IzvorAdap(IzvorSpoj b) : base(b.startJunction.Indeks,
b.endJunction.Indeks, b.napon) {}
	}
	public class OtpornikAdap : LinearnoRKruga.Otpornik {
		Grana otpornik;
		public OtpornikAdap(Grana br) : base(br.startJunction.Indeks,
br.endJunction.Indeks, br.otpor) {
			this.otpornik = br;
			if (br as SklopkaSpoj ) {
				SklopkaSpoj sw = br as SklopkaSpoj;
				if (!sw.Ugasen()) {
					Resistance    = float.MaxValue;
					Conductance = 0.0f;
				} else {Resistance = br.otpor;}
			}			}

	public void PrimijeniRjesenje(LinearnoRKruga.Krug kr) {
			otpornik.struja = kr.GetCurrent(this);
			otpornik.PadNapona = kr.GetVoltage(this);}}
```

Skriptna datoteka 20. RijesiKrug 1. dio.

Funkcija *Primijeni*() ima ulogu održavanja popisa elemenata kruga, popise izvora, otpornika i vodiča. Koristeći popise i funkcije iz sljedeće skripte funkcija računa padove napona, otpore i struje grana i elemenata. Funkcija se poziva svakom promjenom pozicije spoja ili prelaskom pokazivača miša preko elemenata sklopke i vodiča, s obzirom da su to elementi čijom manipulacijom se može

promijeniti stanje strujnog kruga. Po završetku zadatka funkcije šalje se obavijest ostalim sustavima za rješavanje kruga.

```
public override void Primijeni() {
	List<IzvorAdap> izvori = new List<IzvorAdap>();
	List<OtpornikAdap> otpornici = new List<OtpornikAdap>();

	foreach (Grana grana in StrujniKrug.Instance.LG) {
		if (grana is IzvorSpoj) {
			IzvorSpoj izvor = grana as IzvorSpoj;
			izvori.Add(new IzvorAdap(izvor));
			} else if (grana is OtpornikSpoj) {
				otpornici.Add(new OtpornikAdap(grana));
			} else if (grana is VodicSpoj) {
				otpornici.Add(new OtpornikAdap(grana));
			} else if (grana is SklopkaSpoj) {
				otpornici.Add(new OtpornikAdap(grana));

			} else if (grana is TrosiloSpoj) {
				otpornici.Add(new OtpornikAdap(grana));}
		}

		List<LinearnoRKruga.Izvor> izvorList = new List<LinearnoRKruga.Izvor>();
		foreach (IzvorAdap ia in izvori) {
			LinearnoRKruga.Izvor b = ia as LinearnoRKruga.Izvor;
			izvorList.Add(b);}

		List<LinearnoRKruga.Otpornik>		otpornikList		=		new
List<LinearnoRKruga.Otpornik>();
		foreach (OtpornikAdap oa in otpornici) {
			LinearnoRKruga.Otpornik r = oa as LinearnoRKruga.Otpornik;
			otpornikList.Add(r);}

		LinearnoRKruga.Krug	racunajKrug	=	new	LinearnoRKruga.Krug(izvorList,
otpornikList, new List<LinearnoRKruga.IzvorStruje>());
		racunajKrug.Solve();

		foreach (OtpornikAdap ra in otpornici) {
			ra.PrimijeniRjesenje(racunajKrug);
		}
		FKrugRijesen();}}
```

Skriptna datoteka 21. RijesiKrug 2. dio.

Skripta *LinearnoRKruga*() koristi popise ili liste svih elemenata koji su dosad popisani pomoću ostalih funkcija, pamti njihove vrijednosti veličina i spojeve. U nastavku skripte vide se mnoge petlje i brojači koje zapravo računaju vrijednosti struje koristeći zadane vrijednosti napona svih izvora i otpora svakog elementa strujnog kruga. Želi li se prilagoditi metoda izračuna struje unutar strujnog kruga učinit će se to u ovoj skripti. Unutar programskog koda mogu se primijetiti i sustavi sigurnosti u slučaju kratkog spoja ili slijepe grane strujnog kruga. Sama skripta je najduža skripta u radu, izgleda složeno iako zapravo nije, spoj je svih funkcija definiranih u svim prijašnjim skriptama i koristi sve dosad izrađene funkcije kako bi pronašla rješenje strujnog kruga.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

namespace LinearnoRKruga {

	public class Element {
		public int node0;
		public int node1;

		protected Element(int n0, int n1) {
			this.node0 = n0;
			this.node1 = n1;}

		public bool ContainsNode(int n) {
			return n == node0 || n == node1;}
	}
		public class Izvor : Element {
		public float Napon { get; set; }
		public Izvor(int n0, int n1, float v) : base(n0, n1) {
			Napon = v;
			Spoj j0 = StrujniKrug.Instance.LS[n0];
			Spoj j1 = StrujniKrug.Instance.LS[n1];

			List<Spoj> samePosJunctions0 = new List<Spoj>();
			StrujniKrug.Instance.nadjiSpojeveIstePozicije(j0, samePosJunctions0);
			List<Spoj> samePosJunctions1 = new List<Spoj>();
			StrujniKrug.Instance.nadjiSpojeveIstePozicije(j1, samePosJunctions1);

			if (samePosJunctions0.Count > 0 && samePosJunctions1.Count > 0) {
				Napon = v;
			} else {Napon = 0.0f;}
		}}
		public class Otpornik : Element {
		public float Resistance { get; set; }
		public float Conductance { get; set; }

		public Otpornik(int n0, int n1, float r) : base(n0, n1) {
			Resistance = r;
			Spoj j0 = StrujniKrug.Instance.LS[n0];
			Spoj j1 = StrujniKrug.Instance.LS[n1];

			List<Spoj> samePosJunctions0 = new List<Spoj>();
			StrujniKrug.Instance.nadjiSpojeveIstePozicije(j0, samePosJunctions0);
			List<Spoj> samePosJunctions1 = new List<Spoj>();
			StrujniKrug.Instance.nadjiSpojeveIstePozicije(j1,   samePosJunctions1);

			if (samePosJunctions0.Count > 0 && samePosJunctions1.Count > 0) {
				Conductance = 1.0f / r;
			} else {Conductance = 0.0f;}
		}}
		public class IzvorStruje : Element {
		public float Current { get; set; }

		IzvorStruje(int n0, int n1, float i) : base(n0, n1) {
			Current = i;}
	}
```

Skriptna datoteka 22. LinearnoRKruga 1. dio.

```
	public class Krug {
		List<Izvor>		batteries;

		List<Otpornik>	resistors;

		List<IzvorStruje>	currentSources;

		List<int>	nodes = new List<int>();
		int	numNodes;

	Dictionary<int, float>	nodevoltages = new Dictionary<int, float>();

	Dictionary<Element, float> branchCurrents = new Dictionary<Element, float>();

	public Krug(List<Izvor> batteryList, List<Otpornik> resistorList, List<IzvorStruje>
currentSourceList)
			{
			this.batteries		= batteryList;
			this.resistors		= resistorList;
			this.currentSources = currentSourceList;
			SetUpNodes();}

		int GetNumVars() {
			return GetCalcNodeCount() + GetCurrentCount();}

		void SetUpNodes() {
			List<int> temps = new List<int>();
			foreach (Spoj j in StrujniKrug.Instance.LS) {
				bool redundant = false;
				foreach (Spoj sj in j.IstiSp) {
					foreach (int n in temps) {
						if (sj.Indeks == n) {
							redundant = true;}
					}}
				if (redundant)
					continue;
				if (j.spojiSpojeve()) {
					temps.Add(j.Indeks);}
			}
			nodes.Add(StrujniKrug.Instance.LS.Count);
			numNodes = 1;

			foreach (int n in temps) {
				nodes.Add(n);
				++numNodes;}
		}
		int GetCalcNodeCount() {
			return numNodes;}
		int GetCurrentCount() {
			int zeroResistors = 0;
			foreach (Otpornik r in resistors) {
				if (r.Resistance == 0) {
					++zeroResistors;}
			}
			return batteries.Count + zeroResistors;
}
```

Skriptna datoteka 23. LinearnoRKruga 2. dio.

```
float SumConductance(int nodeIndex) {
	float sum = 0.0f;
	foreach (Otpornik rs in resistors) {

		if (rs.ContainsNode(nodes[nodeIndex])) {
			sum += rs.Conductance;
		} else {
			Spoj j = StrujniKrug.Instance.LS[nodes[nodeIndex]];
			foreach (Spoj sj in j.IstiSp) {
				if (rs.ContainsNode(sj.Indeks)) {
					sum += rs.Conductance;}}}
	}
	if (float.IsNaN(sum)) {
		sum = 0.0f;}
	return sum;}
float GetConductance(int node1, int node2) {
	Otpornik result = null;
	foreach (Otpornik rs in resistors) {
		bool sameNodeCandi = false;
		if (rs.ContainsNode(nodes[node1])) {
			sameNodeCandi = true;}
		Spoj j1 = StrujniKrug.Instance.LS[nodes[node1]];
		foreach (Spoj sj in j1.IstiSp) {
			if (rs.ContainsNode(sj.Indeks)) {
				sameNodeCandi = true;}
		}
		if (sameNodeCandi){
			if (rs.ContainsNode(nodes[node2])) {
				result = rs;
				break;}
			else {
				Spoj j2 = StrujniKrug.Instance.LS[nodes[node2]];
				foreach (Spoj sj in j2.IstiSp) {
					if (rs.ContainsNode(sj.Indeks)) {
						result = rs;
						break;}
				}
				if (result != null)
					break;}}
		}
	if (result != null) {
		return result.Conductance;}
	return 0.0f;		}
float SumIncomingCurrents(int nodeIndex) {
	float sum = 0.0f;
	for (int i = 0; i < currentSources.Count; ++i) {
		IzvorStruje cs = currentSources[i];
		if (cs.node1 == nodes[nodeIndex]) {
			sum += cs.Current;
		} else {
			Spoj j = StrujniKrug.Instance.LS[nodes[nodeIndex]];
			foreach (Spoj sj in j.IstiSp) {
				if (cs.node1 == sj.Indeks) {
					sum += cs.Current;}
			}}}
	return sum;}
```

Skriptna datoteka 24. LinearnoRKruga 3. dio.

```
Matrica GetGMatrix() {
       int n = GetCalcNodeCount() - 1;
       Matrica G = new Matrica(n, n);
       for (int i = 1; i < GetCalcNodeCount(); ++i) {
              G[i-1, i-1] = SumConductance(i);
       }

       for (int i = 1; i < GetCalcNodeCount(); ++i) {
              for (int k = 1; k < GetCalcNodeCount(); ++k) {
                     if (i != k) {
                            G[i-1, k-1] = -GetConductance(i, k);}
              }
       }
       return G;}

Matrica GetBMatrix() {
       int n = GetCalcNodeCount() - 1;
       int m = batteries.Count;
       Matrica B = new Matrica(n, m);
 for (int i = 0; i < n; ++i) {
     for (int k = 0; k < m; ++k) {
         float v = 0.0f;
         if (batteries[k].node0 == nodes[i + 1]) {
                            v = 1.0f; }
         else if (batteries[k].node1 == nodes[i + 1]) {
                     v = -1.0f;
         } else {Spoj j = StrujniKrug.Instance.LS[nodes[i + 1]];
                     foreach (Spoj sj in j.IstiSp) {
                       if (batteries[k].node0 == sj.Indeks) {
                                          v = 1.0f;}
                       else if ( batteries[k].node1 == sj.Indeks) {
                                    v = -1.0f;}
                          }
                     }
         B[i, k] = v;}
 }
       return B;}
```

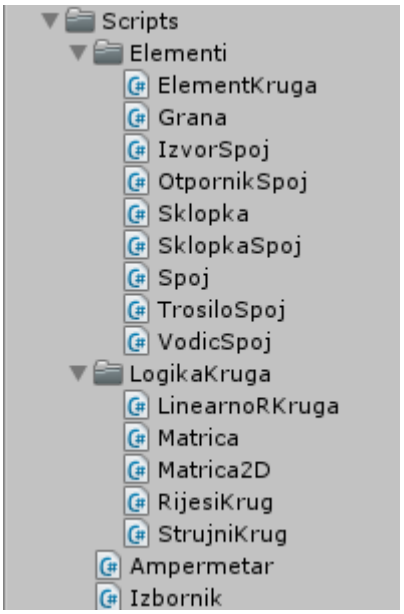
Skriptna datoteka 25. LinearnoRKruga 4. dio.

U određenim dijelovima skripte vidljiv je programski kod za račun vodljivosti umjesto računa otpora. U određenim slučajevima je matematički račun time olakšan, iako je, u posljednjim stadijima izrade rada, dio programskog koda nepotrebno ostao unutar određenih složenijih skripti. Uklanjanje tog nepotrebnog programskog koda stvara više dodatnih problema pa ga je lakše ostaviti unutar skripti. Veoma je bitno pravilno isplanirati tijek rada pri programiranju i velik naglasak treba staviti na sustav kontrole inačica, raditi sigurnosne kopije redovito i dijeliti grane programa u slučaju velikih izmjena.

```
        public bool Solve() {
                if (GetCalcNodeCount() < 1) {return false;}
                        nodevoltages.Clear();
                        branchCurrents.Clear();
                        int n = GetCalcNodeCount() - 1;
                        int m = GetCurrentCount();
                        Matrica A = new Matrica(n+m, n+m);
                        Matrica z = new Matrica(n+m, 1);
                        Matrica G = GetGMatrix();
                        Matrica B = GetBMatrix();
                        Matrica C = Matrica.Transpose(B);

                        bool valid = true;
                        for (int i = 0; i < G.redak; ++i) {
                                if (G[i, i] == 0.0f) {
                                        valid = false;}}
                        if (valid) {
                                valid = false;
                                for (int i = 0; i < B.redak; ++i) {
                                        for (int j = 0; j < B.stupac; ++j) {
                                                if (B[i, j] != 0.0f) {
                                                        valid = true;}}}}
                        if (!valid) {return false;}

                        SetSubMatrix(A, 0, 0, G);
                        SetSubMatrix(A, 0, n, B);
                        SetSubMatrix(A, n, 0, C);

                        for (int i = 1; i < n; ++i) {z[i - 1, 0] = SumIncomingCurrents(i);}
                        for (int i = n; i < n + m; ++i) {z[i, 0] = batteries[i - n].Napon;}

                        Matrica x = A.SolveWith(z);
                        if (x.JeNula()) {return false;}
                        nodevoltages.Add(nodes[0], 0.0f);
                        for (int i = 1; i < GetCalcNodeCount(); ++i) {
                                nodevoltages.Add(nodes[i], x[i - 1, 0]);
                                Spoj j = StrujniKrug.Instance.LS[nodes[i]];
                                foreach (Spoj sj in j.IstiSp) {
                                        nodevoltages.Add (sj.Indeks, x[i - 1, 0]);}}
                        for (int i = 0; i < batteries.Count; ++i) {
                        branchCurrents.Add(batteries[i], -x[GetCalcNodeCount() + i - 1, 0]);}
                        return true;}
                public float GetVoltage(LinearnoRKruga.Element element) {
                        if (nodevoltages.Count < 2 || !nodevoltages.ContainsKey(element.node0)
|| !nodevoltages.ContainsKey(element.node1)) {return 0.0f;}
                        return nodevoltages[element.node1] - nodevoltages[element.node0];}
                public float GetCurrent(LinearnoRKruga.Element element) {
                        if (branchCurrents.ContainsKey(element)) {
                                return branchCurrents[element];}
                        if (element is LinearnoRKruga.Otpornik) {
                                LinearnoRKruga.Otpornik r = element as LinearnoRKruga.Otpornik;
                                return -GetVoltage(r) / r.Resistance;}
                        return 0.0f;}

                static void SetSubMatrix(Matrica A, int row, int col, Matrica B) {
                        for ( int i = 0; i < B.redak; i++ ) {
                             for ( int k = 0; k < B.stupac; k++ ) {
                                        A[row + i, col + k] = B[i, k];}
                }       }       }       }
```

Skriptna datoteka 26. LinearnoRKruga 5. dio.

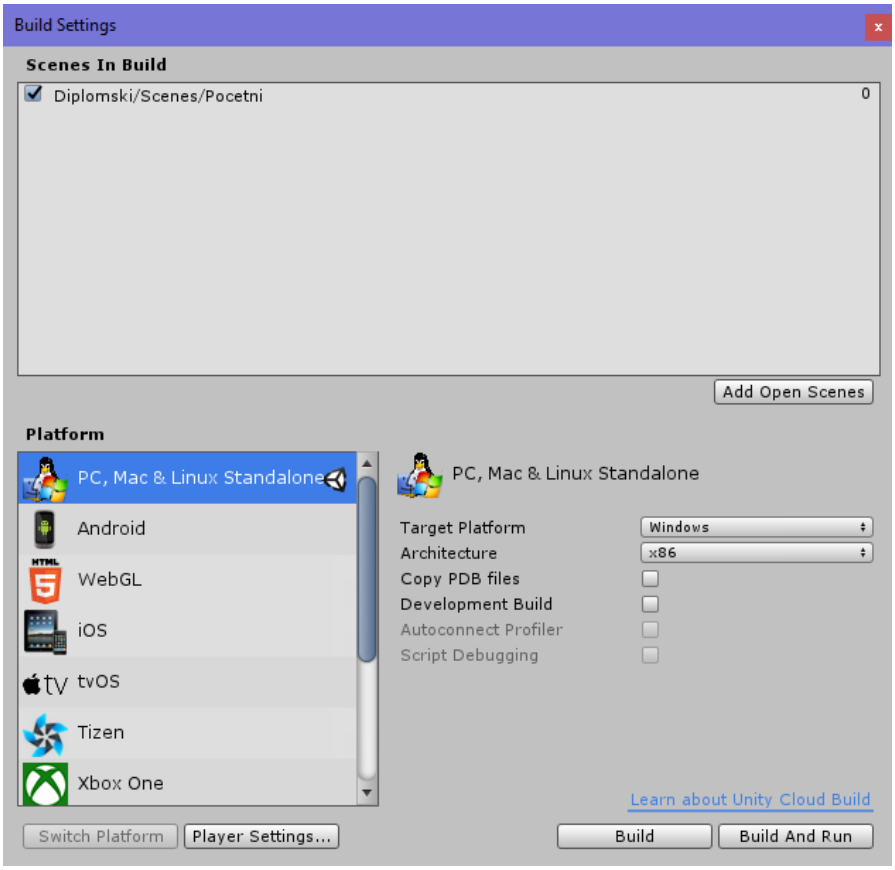
U slučaju eventualnih grešaka pri praćenju i radu slijedi popis svih skriptnih datoteka (Slika 16) potrebnih za rad ovog softvera, njihov sadržaj prikazan je u prethodnim poglavljima. Eventualne greške moguće je riješiti pregledom konzole u donjem prozoru programa *Unity* te pregledom programskog koda u potrazi za greškama u sintaksi odnosno normama programiranja. Greške u sintaksi programskog jezika C# najlakše je ukloniti pregledom programskog koda u Visual Studio 2017 programu. Veoma često greške su odmah pronađene i obilježene, ponekad za iste je ponuđeno rješenje ili primjer ispravne primjene.



Slika 16. Popis svih kodnih skripti rada.

4.7. Završni koraci izrade

U bilo kojem trenutku moguće je testirati izrađeno klikom na gumb sa simbolom trokuta u gornjem dijelu alatne trake. Nakon svih koraka izrade, softver je potrebno složiti u izvršnu datoteku. U glavnom izborniku pri vrhu potrebno je kliknuti na *File* te *Build Settings* kako bi se otvorio prozor s mogućnostima za izradu izvršne datoteke.



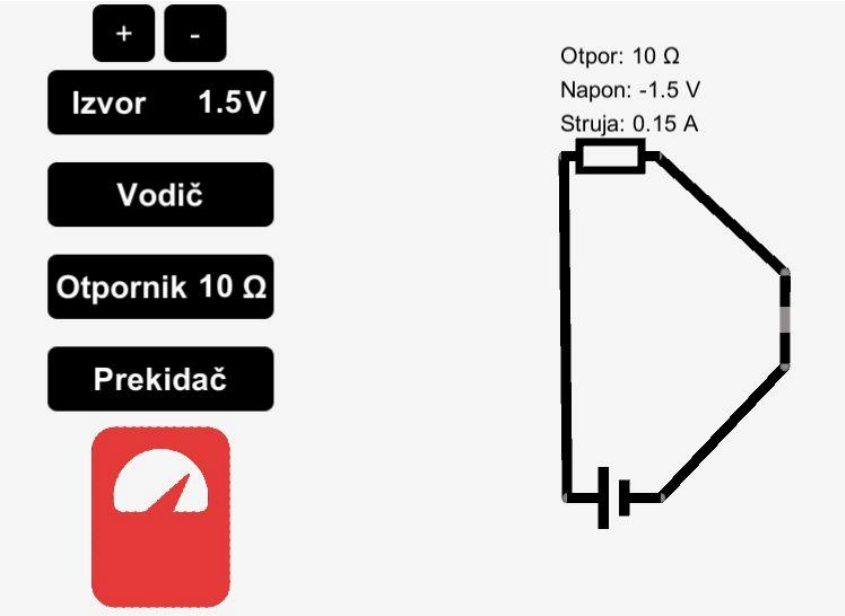
Slika 17. Izbornik za izgradnju izvršne datoteke.

U glavnom prozoru biraju se scene koje je moguće unijeti u rad. U slučaju da je potrebno unaprijed pripremiti više radnih okruženja, primjerice jednu za osnovnu i jednu za srednju školu, moguće je stvoriti više scena i postaviti ih po želji. U ovom radu izabrana je samo jedna jedina scena s kojom je rad i započeo. Zatim je potrebno odabrati platformu za koju se radi izvršna datoteka. Ponudu platformi moguće je proširiti pomoću službenih dodataka *Unityju*. Potrebno je odabrati samostalnu datoteku za *PC*, *Mac* i *Linux*[10] te u pomoćnom desnom izborniku odabrati *Windows*. Sljedeći izbornici daju na biranje arhitekturu, 32-bit i 64-bit, te dodatne mogućnosti u svrhu provjere kvalitete i testiranja rada. Klikom na gumb *Build* otvara se prozor u kojem se bira mjesto na lokalnom spremniku gdje će biti sačuvana izvršna datoteka te je potrebno unijeti ime datoteke.

[10] Računala opremljena jednim od tri najzastupljenija operativna sustava Windows, MacOS i Linux

5. Upute korištenja programa simulacije strujnog kruga

Pokretanjem izvršne datoteke programa otvara se prozor s mogućnostima prije pokretanja samog softvera. Program traži pregled grafičkih mogućnosti i upravljačkih kontrola igrača, zbog korijena programa *Unity* kao platforme za izradu računalnih igara. Karticu kontrola nije potrebno optimizirati jer je upravljanje programom strujnog kruga potpuno pomoću računalnog miša. U kartici grafičkih mogućnosti moguće je odabrati razlučivost prikaza, kvalitetu grafičkog iscrtavanja, zaslon na kojem se prikazuje program i mogućnost odabira između prikaza u prozoru i potpunom prekrivanju zaslona, odnosno *Windowed* označen i neoznačen kvačicom. Preporuka je maknuti kvačicu pokraj *Windowed* radi lakše preglednosti tijekom rada. Klikom na tipku *Play* pri dnu, nakon kratkog uvodnog prikaza, ulazimo u program.



Slika 18. Završeni program strujnog kruga.

S lijeve strane vidljivi su gumbi glavnog izbornika ispod kojih se nalazi dio sučelja u obliku ampermetra. Lijevim klikom na gumb stvara se element koji je naveden na samom gumbu. Element se stvara u središtu prikaza te ga je potrebno pomaknuti povlačenjem elementa za središnji dio sličice elementa. Povlačenjem za krajnji dio elementa, spojni dio u obliku sfere, rotiramo element. Element vodiča je bitno istaknuti jer je upravljanje njime posebno. Klikom na spoj isti se pomiče umjesto rotiranja, stoga se vodič pomiče premještanjem oba spojna dijela. Elementi izvora i otpornika uz naziv elementa imaju i brojčanu vrijednost na gumbu, uz mjernu jedinicu, koja se može mijenjati klikom na broj i upisivanjem željenog broja kao nove vrijednosti. Elementi se postavljaju na polje proizvoljno, spajaju se elementima vodiča povlačenjem spojeva jednih preko drugih, kad su spojevi blizu samostalno se zalijepe jedni za druge. Spojeve je također moguće i

odvojiti te promijeniti konfiguraciju strujnog kruga. Ampermetar u korisničkom sučelju služi kao pokazatelj toka struje, odnosno ukoliko je spojen pravilan strujni krug ampermetar mijenja boju u crvenu kako bi naznačio da je strujni krug zatvoren. Desnim klikom na otpornike strujnog kruga stvara se mali izbornik iznad elementa koji očitava pad napona, struju i otpor odabranog otpornika. Element prekidača potrebno je zatvoriti klikom na sklopku istaknutu drugom bojom, element je također preporučljivo koristiti jer sadrži najviše provjera za ažuriranje stanja kruga.

5.1 Pripremljeni sat

U prilogu se nalazi priprema jednog nastavnog sata u osnovnoj školi, a u nastavku slijede upute za prilagodbu programa strujnog kruga za taj sat. Tema nastavnog sata je Ohmov zakon i zamisao je izvesti demonstraciju ili pokus kojim je prikazana linearna ovisnost jakosti struje o promjeni napona u strujnom krugu te time uvesti električni otpor. Prva promjena koju je potrebno napraviti u programu je stvaranje nove scene odabirom New Scene u glavnom izborniku datoteka. Novonastalu scenu potrebno je popuniti svim objektima iz prvotne scene jednostavnim kopiranjem i lijepljenjem svih objekata. Svaka scena bit će spremljena u mapu *Scenes* gdje je moguće otvoriti svaku scenu pojedinačno. Desnim klikom na otpornik prikazan je otpor otpornika što je kontraproduktivno u predviđenom pripremljenom nastavnom satu. Stoga je potrebno ukloniti prikaz otpora pri desnom kliku na otpornik. To se čini pretvaranjem jednog retka programskog koda u komentar[11], dodavanjem dvije kose crte u skripti *Ampermetar* sljedećeg retka:

```
	private void IzmjeriOtpor(int windowID) {
		guiStyle.fontSize = 20;
		//GUI.Label(rRect, "Otpor: " + otpor.ToString() + " Ω", guiStyle);
		GUI.Label(vRect, "Napon: " + napon.ToString() + " V", guiStyle);
		GUI.Label(iRect, "Struja: " + struja.ToString() + " A", guiStyle);}
```

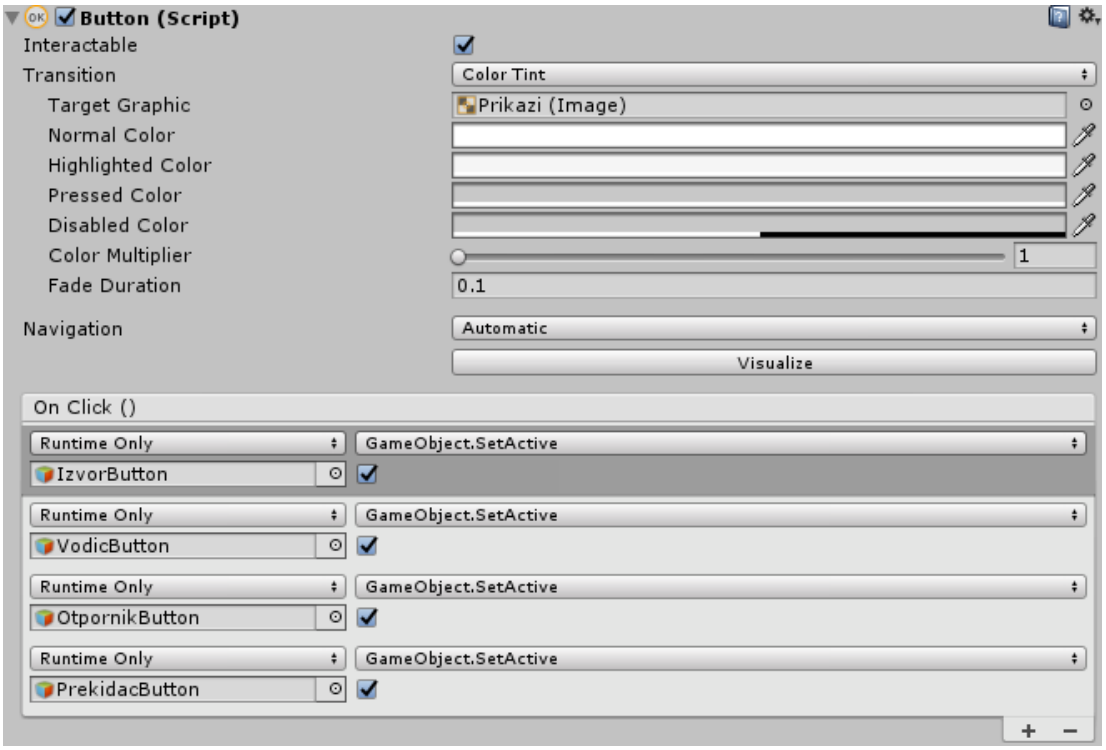
Za nastavni sat je predviđena izvedba dva seta mjerenja. Potrebno je pripremiti dva različita otpornika dupliciranjem predloška elementa otpornika, njegovim preimenovanjem u *Otpornik2* te promjenom izgleda pomicanjem trodimenzionalnih elemenata predloška istog otpornika. Gumb za stvaranje otpornika potrebno je razdijeliti na dva gumba, imenovana *Prvi* i *Drugi*. Zatim se u skriptnoj datoteci *Izbornik* duplicira programski kod otpornika i učine sljedeće promjene:

```
public void StvoriOtpornik2()
    {   GameObject obj = Instantiate(gotoviOtpornik2) as GameObject;
        OtpornikSpoj rNode = obj.GetComponent<OtpornikSpoj>();
        if (rNode != null)
        {
            rNode.otpor = otpor - 5;}}
```

[11] Kod ili tekst u obliku komentara zanemaren je pri očitavanju naredbi

Početna postavka otpora otpornika jest deset ohma, u novom otporniku pet ohma postiže se najjednostavnije oduzimanjem pet od početne vrijednosti. Nakon ovog dodatka potrebno je predlošku povući u predviđeno polje izbornika u prozoru hijerarhije te funkciju *StvoriOtpornik2* odabrati u izvršnoj skripti predviđenog gumba.

Koristan dodatak bili bi gumbi za skrivanje i prikazivanje sučelja izbornika kako bi pažnju učenika zadržali na spoju i mjerenjima, a ne na elementima i simbolima koje ne prepoznaju. Pomoću izbornika *GameObject – UI – Button* dodaje se gumb. U pregledniku se mijenja tekst gumba u znak plus, boja gumba, boja teksta te pri dnu preglednika u kartici *Button (Script)* (Slika 19.) klikom na znak plus trake *On Click ()* dodaju se četiri kartice. U donji redak kartica se povlače gumbi izbornika iz prozora hijerarhije i time se daju upute programu da na klik gumba izvrši zadanu radnju na umetnutom objektu. Radnje se zadaju klikom na redak *No Function* i u padajućem izvorniku se bira *GameObject – SetActive (bool*). Kvadratići pokraj izabranih objekata aktiviraju se kvačicama. Na klik gumba, zadani objekti se sada aktiviraju, što trenutno ne čini ništa. Gumb se duplicira desnim klikom i odabirom *Duplicate*, duplikat se pomakne, preimenuje i tekst promijeni u minus. Kvačice u kućicama pokraj odabranih objekata u kartici *On Click ()* se ukloni čime se zadana radnja mijenja i objekt se deaktivira. Time su izrađeni gumbi za aktivaciju i deaktivaciju izbornika.



Slika 19. Kartica *Button(Script)* u pregledniku gumba za prikaz izbornika.

6. Zaključak

Unatoč tome što je tema rada izrada softvera za simulaciju strujnog kruga kojim bi se zamijenio rad sa stvarnim kompletima strujnog kruga, autor i dalje smatra da je praktični rad na nastavi najbolji pristup radu s učenicima te da takav pruža učenicima iskustva koja je teško nadomjestiti bilo kakvim simulacijama. Softver stoga može biti korišten kao zamjena za rad u nastavi uvjetovana nedostatkom materijalnih sredstava, potrebe za individualnim radom ili radom u paru ili primjerice potrebom za sigurnom okolinom u kojoj učenik može bez straha od posljedica istraživati. Jednako tako, moguće je praktični rad sa strujnim krugovima miješati s radom pomoću simulacija i na taj način pojačati asocijaciju realnih elemenata strujnih krugova s apstraktnim simbolima shematskog prikaza.

Prilikom primjene na probnim satima nastave zanimljivo je istaknuti nekolicinu opažanja. Prvo opažanje usmjereno je na manjak straha od strane učenika pri eksperimentiranju s vrijednostima napona i otpora elemenata strujnog kruga. Određena skupina učenika s namjerom je tražila vrijednosti koje će zaustaviti ili usporiti simulaciju. Njihovu znatiželju moguće je nagraditi razgovorom o visokim naponima i otporima u stvarnom životu. Zanimljivo je primijetiti manjak problema tijekom rada s vrijednostima struje ili napona, niti u jednom trenutku nije bilo potrebe objašnjavati odstupanja mjerenja pojedinog učenika spram drugih učenika. U radu je korištena pametna ploča, odnosno velika ploha za prikazivanje slike projektora s mogućnostima dodirnog upravljanja. Korištenje softvera time je svedeno na uporabu koja je po motoričkim radnjama gotovo identična korištenju softvera na tabletu ili pametnom telefonu koje je koristila većina učenika. Interes učenika nije pao nakon prvog korištenja softvera te su određeni učenici softver koristili i kod kuće na svojim uređajima.

Bitno je istaknuti i probleme tijekom postavljanja i korištenja softvera. Određene inačice operativnih sustava i uređaja nisu prihvaćale instalaciju softvera. Ponekad dodirne kontrole nisu bile dostupne na uređaju, čineći time program neuporabljiv. Razlučivost prikaza također je u nekolicini slučajeva predstavljala problem jer bi „odsjekla“ dio prikaza softvera i onemogućila daljnju uporabu. Unatoč problemima, korištenje softvera omogućilo je samostalan rad većeg broja učenika od uobičajenog iz jednostavnog razloga dostupnosti opreme. Uz bolju razradu teme i određenih unaprijeđena korisnost softvera zasigurno bi još porasla. Unaprjeđenje softvera jedan je od razloga zbog kojeg je softver izrađen u trenutnom obliku osnovnog okvira te su u nastavku iznesene neke od mogućnosti dodataka i poboljšanja.

Mogućnost spremanja i učitavanja izgrađenih strujnih krugova zahtijevala bi sustav pamćenja elemenata, njihovih koordinata i pohrane tih podataka u nekom obliku. Pogodnosti u obliku

pripreme za određene jedinice, pripremljenih zadataka za vježbu i provjeru te opća ušteda vremena u ponavljanju vježbi i spojeva bile bi neizmjerne.

Ukoliko je odabrana jednostavna estetika shematskog spoja, crni elementi na bijeloj pozadini, koji prate međunarodni standard dizajna elemenata, moguća nadogradnja bila bi snimanje i spremanje strujnih krugova u obliku digitalnih slika. Slike bi se zatim mogle iskoristiti u prezentacijskim materijalima, pisanim vježbama i provjerama znanja.

Jedan propust trenutne inačice programa je nemogućnost brisanja elemenata strujnog kruga nakon što su stvoreni pomoću izbornika. Brisanje elemenata nije problematično uvesti, no trenutni algoritam rješavanja kruga nije dovoljno fleksibilan za sortiranje praznih mjesta koja ostaju nakon uklanjanja elemenata.

Elementi strujnog kruga kao što su: trošilo, kondenzatori, promjenjivi otpornici, vodiči različitih dimenzija ili materijala bili bi odlični dodaci kojima bi omogućili korištenje softvera na daleko većem broju jedinica nastavne cjeline elektromagnetizma. Pri izradi trošila bilo bi zanimljivo izraditi element koji će mijenjati razinu svijetla s promjenom struje koja prolazi kroz element. Za dodatak promjenjivih otpornika i kondenzatora bilo bi potrebno napraviti promjene algoritma strujnog kruga s obzirom na njegov trenutni način rada. Drugim riječima, dodaci nisu jednostavni za izraditi, ali su mogući. Također, račun snage i potrošnje električne energije elemenata lako se mogu dodati i time obuhvatiti još nastavnih jedinica.

Naposljetku, koristeći mogućnost scena ugrađenih u program *Unity*, program je moguće proširiti i ostalim cjelinama nastavnog programa fizike, gradeći postupno jedan program za korištenje tijekom cijele nastavne godine. Jednako zanimljivo bilo bi proširenje u obliku kvizova, provjera i interaktivnih igara na temelju izrađenih simulacija gdje bi se na zabavan način moglo iskoristiti učeničko poznavanje softvera i ispitati ili ponoviti znanje obrađenog gradiva.

Korištenje računalnih simulacija u koraku je s vremenom i postaje veoma bitan dio alata kojim učitelj raspolaže. Samostalni razvoj softvera za nastavu daje veliku razinu interaktivnosti nastavnim procesom i, uz trud, povećava kvalitetu cjelokupne nastave.

7. Literatura

1) Engelhardt P., Beichner R. J. : Students` understanding of direct current resistive electrical circuits : American journal of physics 72, (2004.)
2) Jackson Simon : Unity 3D UI Essentials : Packt Publishing, Birmingham, 2015.
3) Jurić Ana: Diplomski rad :Računalna implementacija konceptualnog testa iz jednostavnih strujnih krugova : Fizički odsjek, Prirodoslovno – matematički fakultet, Sveučilište u Zagrebu, Zagreb, 2006.
4) Labor Jakov, Fizika 2, udžbenik za drugi razred gimnazije, Alfa d.d., Zagreb, 2014.
5) Moore J. Cristhopher: Efficacy of multimedia learning modules as preparation for lecture-based tutorials in electromagnetism : Department of Chemistry and Physics, Coastal Carolina University, Conway, 2014.
6) Muller Derek Alexander: Designing Effective Multimedia for Physics Education : School of Physics, University of Sydney, Sydney, 2008.
7) Okita Alex : Learning C# Programming with Unity 3D : A K Peters/CRC Press, SAD, 2014.
8) Paar Vladimir, Fizika 2, udžbenik za drugi razred gimnazije, Školska knjiga, Zagreb, 2005.
9) Paar Vladimir, Klaić Mladen, Sila Dubravko, Ćulibrk Tanja, Martinko Sanja : Fizika oko nas 8 : udžbenik za osmi razred osnovne škole, Školska knjiga, Zagreb, 2013.
10) Prelovšek Peroš Sonja, Mikuličić Branka, Milotić Branka, Aviani Ivica : Otkrivamo fiziku 8 : udžbenik fizike s višemedijskim nastavnim materijalima u osmom razredu osnovne škole : Školska knjiga : Zagreb, 2013.

Internet poveznice:

- https://phet.colorado.edu/en/simulation/circuit-construction-kit-dc (11.10.2017.)
- http://www.physicsclassroom.com/Physics-Interactives/Electric-Circuits/Circuit-Builder/Circuit-Builder-Interactive (11.10.2017.)
- http://www.falstad.com/circuit/ (11.10.2017.)
- https://dcaclab.com/ (11.10.2017.)
- https://unity3d.com/ (11.10.2017.)
- https://docs.unity3d.com/ScriptReference/index.html (11.10.2017.)
- https://docs.unity3d.com/Manual/index.html (11.10.2017.)

8. Životopis

Rođen sam 27. listopada 1988. godine u Vinkovcima. Također u Vinkovcima, započinjem obrazovanje u Osnovnoj školi Bartola Kašića te ga nastavljam pohađanjem općeg smjera Gimnazije Matije Antuna Reljkovića. Kao redoviti student upisujem se 2007. godine na preddiplomski studij Odjela za fiziku sveučilišta J.J.Strossmayera u Osijeku koji završavam 2013. godine. Iste godine nastavljam školovanje upisom na diplomski studij Odjela za fiziku nakon kojeg se nadam steći zvanje magistra edukacije fizike i informatike.

9. Prilozi

Škola: OŠ Ivana Kozarca Županja
Mjesto: Županja
Br. sata:
Razred: 8.

Ime i prezime nastavnika: Adrijan Čačić
Nastavno područje: FIZIKA
Broj sati: 1
Tip sata: obrada

Priprema za izvođenje nastavne teme
Ohmov zakon

Cilj nastavne teme:	istražiti međusobnu ovisnost električnog otpora i struje o naponu te iskazati Ohmov zakon		
Obrazovna postignuća:	Učenik će moći: 1. prepoznati i sastaviti jednostavni strujni krug 2. grafički prikazati ovisnost struje o naponu za različite otpore 3. grafički prikazati ovisnost otpora o naponu i utvrditi da ne postoji ovisnost 4. iskazati Ohmov zakon 5. primijeniti Ohmov zakon u računskim zadacima		
Odgojna postignuća:	1. poticati zanimanje za istraživanje 2. stjecati naviku ispravne upotrebe računalne opreme ili pribora pokusa 3. prihvatiti znanstveni način mišljenja i formaliziranog iskazivanja		
Vrednovanje obrazovnih ishoda	Vrednovanje se ostvaruje pitanjima postavljenima tijekom nastavnog procesa, ponavljanju obrađenog i obradi novog gradiva. Također kroz rezultate rješavanja numeričkih zadataka, obavljanju praktičnog rada (mjerenje struje u programu) pri prikazu rezultata (crtanje grafa) i rješavanju zadaće.		
Ključni pojmovi:	Ohmov zakon, struja, napon, otpor, strujni krug	**Korelacija**:	matematika, tehnička kultura

Organizacija nastavnog procesa

Nastavne metode:	• metoda demonstracije pokusa • višesmjerna komunikacija	**Socijalni oblici rada:**	• Frontalni (cijeli nastavni proces) • Individualni (rad s aplikacijom za strujni krug) • Rad u paru (u slučaju manjka opreme pri radu s aplikacijom)
Nastavna sredstva:	Udžbenik, program simulacije strujnog kruga, računalo, projektor	**Nastavna pomagala:**	Ploča, kreda, pametna ploča (ako je dostupna), tableti ili smartfon uređaji, trokuti za skiciranje grafa, listići
Literatura za pripremu nastavnika:	Udžbenik	**Literatura za učenike:**	Udžbenik

Tijek nastavnog procesa

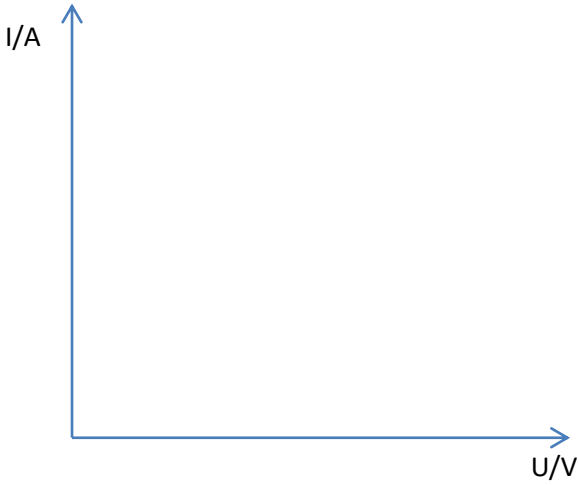
	Aktivnost nastavnika	Aktivnost učenika
Uvodni dio	Učenici su s prethodnih nastavnih jedinica upoznati s definicijama napona i struje. Jedinica prije ove upoznala ih je s konceptom električnog otpora, njegovu ovisnost o fizičkim karakteristikama vodiča. Učenicima predstavljamo cilj sata postavljanjem pitanja kakav odnos imaju napon, struja i otpor. Ponavljanje i motivaciju sata radimo pomoću pitanja: **Što je napon?** **Što je struja?** **Što čini otpor u vodiču?** **Što će se dogoditi sa strujom ako povećamo napon strujnog kruga?** **Što će se dogoditi sa strujom ako povećamo otpor strujnog kruga?** Nakon što učenici uspješno odgovore na pitanja navodimo ih na planiranje pokusa kojim provjeravaju tvrdnje: **Koje veličine mijenjamo i mjerimo?** **Kako će izgledati tablica za unos podataka?** **Kako će izgledati shema strujnog kruga?** **Kako ćemo prikazati rezultate?** Nakon što učenici uspješno odgovore na sva pitanja i postave metode rada, obavijestimo ih o programu koji će koristiti za rad vježbe. Dajemo im upute za postavljanje programa na uređajima koje koriste te dijelimo pripremljene listiće koje će popunjavati tijekom vježbe. Tražimo od učenika da u bilježnicama ostave mjesta za naslov te upišu „Vježba“, te skiciraju shemu strujnog kruga koji će spajati.	Slušaju izlaganje. Odgovaraju na pitanja. ***Napon je energija po jediničnom naboju koja se iz električnog izvora prenosi strujnim krugom.*** ***Struja je količina naboja koja poprečnim presjekom vodiča prođe u jednoj sekundi.*** ***Otpor čini sudaranje nositelja naboja i atoma vodiča.*** ***Struja se povećava povećanjem napona.*** ***Struja se smanjuje povećanjem otpora.*** Sudjeluju u razradi vježbe. ***Mijenjamo napon, mjerimo struju.*** ***Tablica ima stupac za napon, struju i omjer.*** ***Jednostavni strujni krug. Izvor, trošilo, sklopka, vodiči, ampermetar, voltmetar.*** ***Graf.*** Prate upute za pripremu rada. Pregledavaju listiće. Bilježe zadano.
Središnji dio	**Vježba u programu za simulaciju strujnog kruga** Vježbu učenici izvode individualno ili u parovima ovisno o dostupnosti uređaja (pametnih telefona, tableta ili računala). Prije nego učenici pokrenu program, nastavnik ga pokreće sam i pomoću projektora prikazuje i pojašnjava korisničko sučelje. Prolazi redom svaki gumb i pojašnjava njegovu ulogu. Demonstrira stvaranje jednostavnog strujnog kruga, kako se elementi stvaraju, povezuju te kako se vrši mjerenje. Bitno je napomenuti učenicima da izvore stvaraju dok vrše mjerenja zbog ograničenja programa. Primjerice, ako postoje četiri stvorena izvora, svi moraju biti spojeni u strujni krug, u suprotnom strujni krug neće biti zatvoren. Veoma je bitno stoga stvarati novi izvor za serijski spoj tek nakon što je izvršeno mjerenje s prethodnim brojem izvora.	Pripremaju se za rad. Prate objašnjenje korištenja programa. Postavljaju pitanja u slučaju nejasnoća.

	Pri objašnjavanju gumba „Prvi“ nastavnik navodi da on stvara vodič određenog poprečnog presjeka, materijala i duljine, a time i otpora. Gumb drugi preskoči za kasnije. Ampermetar ispod izbornika indikator je zatvorenog kruga. Bitno je obavijestiti učenike da u slučaju greške ponovno pokrenu program i da elemente ne stvaraju nasumično ili prekomjerno jer iste nije moguće obrisati bez ponovnog pokretanja. Bitno je demonstrirati: • Stvaranje elementa lijevim klikom na gumb • Pomicanje elementa lijevim klikom i povlačenjem • Rotiranje elementa lijevim klikom i povlačenjem spoja na krajevima elementa • Spajanje elemenata vodičima povlačenjem krajeva vodiča • Mjerenje struje i napona desnim klikom na vodič koji testiramo • Pravilno spojen krug pomoću crvenog ampermetra Nakon pojašnjenja rada s programom nastavnik upućuje učenike na početak rada. Tijekom rada promatra i pomaže pri radu. Pitanjima provjera napredak rada učenika i pregledava rezultate mjerenja koje učenici upisuju u listiće. Preporuka je provjeriti rezultate prvog mjerenja dosta rano kako bi otkrili velike nepravilnosti u radu. Svi učenici bi trebali imati jednake rezultate. Po završetku mjerenja nastavnik od učenika traži prepoznavanje pravilne veze između napona i struje. Nakon što pravilno prepoznaju linearnu zavisnost napona i struje upućujemo ih u pisanje matematičkog omjera napona i struje i treći stupac tablice u listićima. Pošto je omjer stalan zamijenit ćemo ga simbolom i otkriti da se radi o otporu vodiča: $$R = \frac{U}{I}$$ U slučaju prvog vodiča otpor iznosi 10Ω. Otpor kovinskog vodiča je stalan i neovisan o naponu. Zatim učenicima predložimo ponavljanje mjerenja, ali da vodič zamijenimo s vodičem koji ima duplo veći poprečni presjek. Nastavnik traži od učenika zaključak o promjeni vrijednosti otpora u novom slučaju. Upućujemo učenike da mjerenja ponove s vodičem kojeg stvore gumbom „Drugi“. Popunjavaju stupce 4, 5 i 6 s novim podacima. Nakon završenih mjerenja i zapisa mjerenja, tražimo od učenika da grafički prikažu podatke iz tablica za oba vodiča. Skiciramo graf na ploču kako bi učenici imali povratnu	Otvaraju program na svojim uređajima. Postavljaju pitanja u slučaju nejasnoća. Traže i pružaju pomoć. Spajaju dogovoreni strujni krug i vrše mjerenja. Odgovaraju na pitanja. Zapisuju rezultate mjerenja u listiće. Odgovaraju na pitanja. Ponavljanju vježbu s drugim vodičem. Popunjavaju preostale stupce tablice na listićima. Unose mjerenja na graf. Crtaju graf.

	informaciju o ispravnosti skice. Provjeravamo shvaćaju li učenici graf postavljanjem pitanja: **Gdje bi postavili pravac vodiča otpora 15Ω?** **Što nam govori veći ili manji nagib pravca?** **Možete li odrediti struju za vrijednost napona koju niste izravno izmjerili (npr. 10V)?** Ponavljamo otkrića nastavne jedinice. Otpor je stalan i neovisan o naponu i struji. Štoviše, otpor vodiča i napon na njegovim krajevima diktiraju struju kroz njega. Napišemo li prethodni matematički izraz tako što izrazimo struju dobijemo (dajemo uputu učenicima da popune listić): $$I = \frac{U}{R}$$ Tražimo od učenika da izraz iskažu riječima. Zakon koji po Georg Simon Ohmu nazivamo Ohmovim zakonom. ***Električna struja kroz kovinski vodič stalne temperature razmjerna je električnom naponu, a koeficijent proporcionalnosti (razmjernosti) je recipročna vrijednost električnog otpora.*** Nakon što učenici popune listić podsjetimo ih da upišu naslov „Ohmov zakon“.	***Manji nagib pravca odgovara većem otporu. Veći nagib manjem otporu.*** ***Pravac vodiča 15Ω postavljamo ispod prijašnja dva.*** ***Pratimo okomicu na os napona te u točki sjecišta s pravcem povlačimo okomicu na os struje i očitavamo ju. Odgovara strujama 1A i 2A.*** Slušaju definicuju Ohmovog zakona. Učenici pokušavaju iskazati matematički izraz riječima odnosno kreiraju iskaz Ohmovog zakona. Zapisuju bilješke. Popunjavaju listić.
Završni dio	Ukratko ponavljamo tijek sata i najvažnije zaključke. Učenici zapisuju numerički zadatak koji rješavaju za zadaću ako ga ne stignu riješiti do kraja sata. Zadatak: Koliki je otpor vodiča na čijim krajevima voltmetrom mjerimo napon od 9000 mV, te mjerimo ampermetrom da njime teče struja od 0,03 hA? Za koliko bi promijenili struju kroz vodič kada bi prepolovili napon izvora? Možemo li promijeniti struju kroz vodič bez da mijenjamo napon izvora?	Postavljaju pitanja. Zapisuju i rješavaju zadatak.

Prilog: listići

Broj mjerenja	Napon U/V	Struja I/A	Omjer $\frac{U}{A}$	Napon U/V	Struja I/A	Omjer $\frac{U}{A}$
1.						
2.						
3.						
4.						
5.						
6.						



_____ zakon:

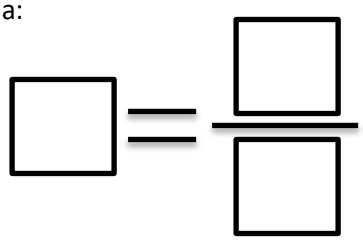
_____

_____

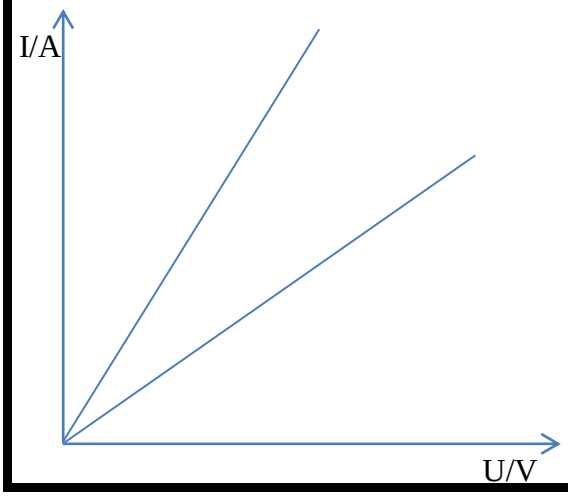
_____

_____

Formula:



PLAN PLOČE/PLAN PROJEKTORA



Zadatak: Koliki je otpor vodiča na čijim krajevima voltmetrom mjerimo napon od 9000 mV, te mjerimo ampermetrom da njime teče struja od 0,03 hA? Za koliko bi promijenili struju kroz vodič kada bi prepolovili napon izvora? Možemo li promijeniti struju kroz vodič bez da mijenjamo napon izvora?