SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3029

VIZUALIZACIJA PROGRAMIRANJA

Joel Mislav Kunst

Zagreb, lipanj 2013

SADRŽAJ

1. Uvod

Programiranje je danas jako raširena aktivnost te se njegova primjena proteže u gotovu svu današnju tehnologiju. Ovaj rad govori o tome kako programiranje približiti djeci rane dobi te objašnjava neke od razloga zašto bi to uopće bilo dobro.

U prvom poglavlju (2.) objašnjava se ukratko što je to programiranje i čemu služi, te se daje neki kratki uvod u vizualno programiranje.

U drugom poglavlju (3.) opisuje se igra izrađena za ovaj rad te kako se ona koristi.

U trećem poglavlju (4.) detaljnije se opisuje realizacije igre, korišteni alati, te problemi na koje se nailazilo prilikom izrade uz njihova rješenja.

Na samom kraju dan je zaključak koji ukratko opisuje saznanja dobivena tokom izrade ovog rada.

Nakon zaključka nalazi se popis literature koja ovdje nije velika iz razloga što se razvijala igra temeljena na vlastitoj ideji, a literatura su samo sitnice koje su pomogle u realizaciji ideje.

Nakon literature nalazi se kratki sadržaj na hrvatskom i engleskom jeziku.

2. Programiranje

2.1. Programiranje

Programiranje je davanje uputa računalu što i kako učiniti. Davanje uputa može se ostvariti pisanjem koda u nekom od programskih jezika. Programiranje je umjetnost i umijeće u stvaranju programa za računala. Stvaranje programa sadrži u sebi pojedine elemente dizajna, umjetnosti, znanosti, matematike kao i inženjeringa. Osoba koja stvara program zove se programer.

Programiranje se također može nazvati i vještinom pomoću koje programer stvara algoritme. Algoritmi su načini ili postupci kojima računalo obavlja neki zadatak, skup koraka u izvođenju programa. Razni zadaci mogu se napraviti na mnogo različitih načina. Za osmišljavanje algoritama potreban je poseban način razmišljanja, poznavanje matematike, vježba i iskustvo. Osim osmišljavanja samih algoritama programer može imati i drugačije zadaće, implementacije raznih njemu poznatih algoritama kako bi riješio neki problem, ali da njegov produkt (kod koji je napisao) bude čitljiv drugima i njemu te da bude dobro dokumentiran. Dizajn samog rješenja problema u smislu organizacije koda bi se moglo nazvati umjetnošću.

2.2. Načini programiranja

Kao što je već ranije spomenuto programiranje se može izvesti pisanjem koda (takozvano kodiranje), ali osim toga postoje razni načini kojima se može napraviti program, te se u širem smislu i to može zvati programiranjem. Razvoj nekakvog programa kroz bilo kakav od načina dijeli sličnu logiku razmišljanja u cilju pronalaska što boljeg rješenja i algoritamski i organizacijski (misli se na organizaciju i čitljivost koda).

Kako bi kod bio što čitljiviji i ne samo čitljiviji i organiziraniji nego kako bi njegova organizacija bila dizajnirana na takav način da se cijeli način razmišljanja programera digne na jednu višu razinu postoje razne apstrakcije kao što je na primjer objektno orijentirana programska paradigma. Klasični programi pisali su se tako da sadrže instrukciju za instrukcijom koje se tako redom izvršavaju, te je način razmišljanja tvorca takvog programa bio upravo takav, morao je razumjeti i osmisliti svaki maleni korak i imati puno različitih ne povezanih podataka na umu jer su svi oni postojali na tom jednom mjestu i slijedno se izvršavali. Kod objektno orijentirane paradigme uveden je pojam objekta gdje se sve to apstrahira tako da sada koliko god da se program izvršava slijedno neke stvari su odvojene u zasebne cjeline i na taj su način „sakrivene“ od programera. Sakrivene u smislu da u tim „skrivenim“ trenucima programer o njima ne treba razmišljati. Kod se sada oblikuje tako da se smislene cjeline grupiraju u nešto što nazivamo objektima. Objektna paradigma ima još mnogo značajki i korisnih detalja no u okviru ovog rada nas zanima nešto drugo.

2.3. Vizualizacija programiranja

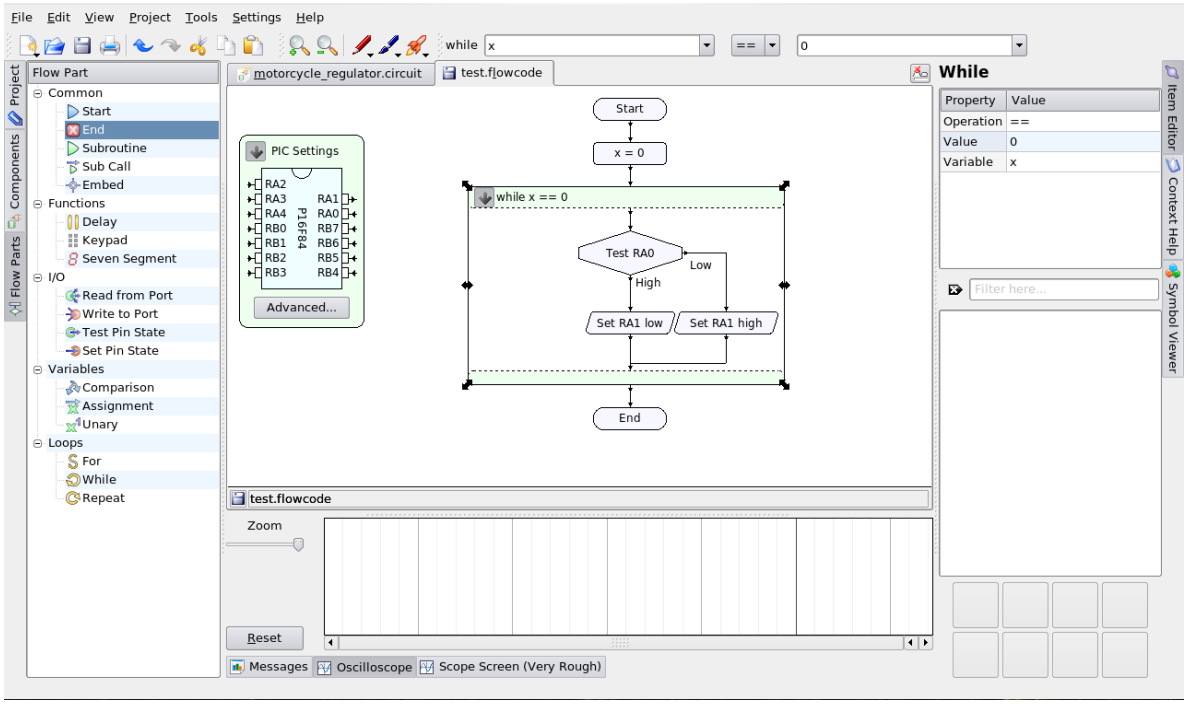
Kao i objektna paradigma osim drugih paradigmi postoje i druge apstrakcije, a jedna od njih je razina na kojoj „programer“ ne treba pisati nikakav kod nego cijeli program može slagati vizualno od ponuđenih komponenata.

Postoje različiti načini vizualnog programiranja. U trenutku pisanja ovog rada alati vizualnog programiranja za izradu programa, aplikacija koje se mogu mjeriti sa onim klasičnima te tako koristiti u svakodnevnoj nisu još toliko razvijeni. Vizualno programiranje može se također koristiti u edukacijske svrhe, a takav je i ovaj rad. Takvi alati mogu koristiti učenju mladih umova

načinu razmišljanja koje će im jednog dana možda trebati. Također takvi se alati mogu koristiti za zabavu uz „trening“ uma.

Vizualnim programiranjem se programeru omogućuje stvaranje programa manipuliranjem elementima grafički umjesto tekstualno. U većini vizualnih programskih jezika se programira na način da se „kutije“ povezuju strelicama. Kutije prikazane geometrijskim oblicima predstavljaju entitete koji se spajaju linijama, strelicama, lukovima... kako bi predstavili nekakvu vezu među entitetima. Danas postoji mnogo alata za vizualno programiranje. Primjer uporabe jednog alata u kojem se programira vizualno prikazan je na *slici 1*.

U projektu napravljenom za ovaj rad korišten je malo drugačiji pristup. Instrukcije programa postoje kao ikonice od kojih se slaže program. Za povezivanje elemenata ne koriste se nikakve linije, strelice, lukovi ili slično, nego se instrukcije slijedno izvode jedna za drugom onako kako su složene.



Slika 1. Alat za vizualno programiranje

3. Igra KockoProgramirač

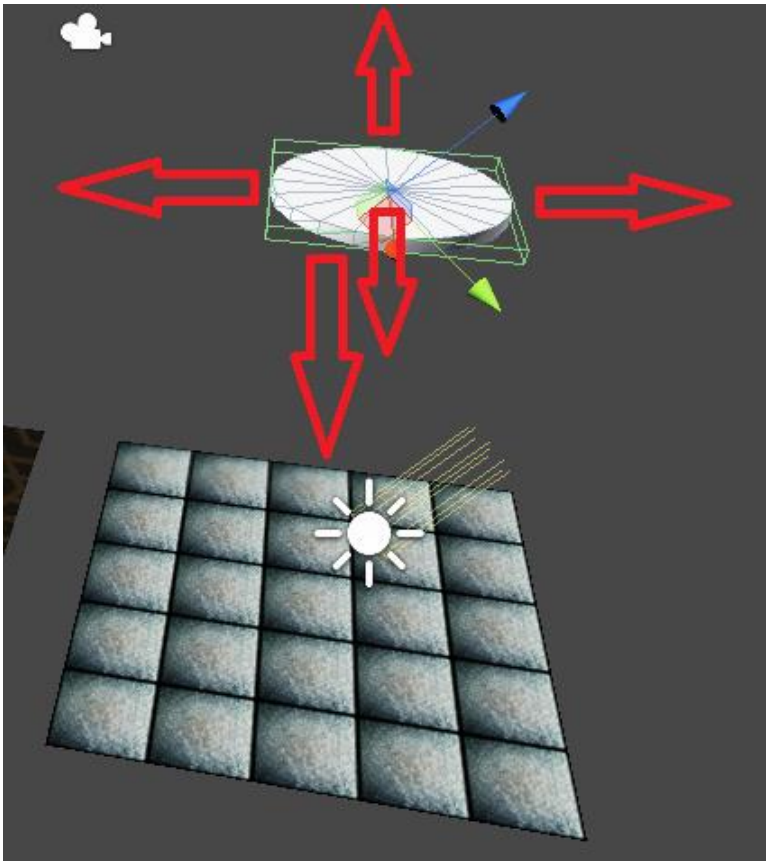
3.1. O aplikaciji

KockoProgramirač je igra koja ima elemente vizualnog programiranja i umjetne inteligencije, te služi kao edukacijski alat za djecu i zabavu za starije uzraste (problemi itekako mogu biti izazovni i za starije uzraste). Radi se o logičkoj igri u kojoj je potrebno presložiti kocke u 3D prostoru. Preslagivanje kocaka ne radi se ručno nego se programira dizalica koja ima mogućnost raznošenja kocaka kako bi kocke iz neke „početne“ konfiguracije kocaka dovela da tražene konfiguracije.

Dizalica je zamišljena kao nekakav magnet za kocke, te ih stoga može privući prema sebi. Dizalica u jednom trenutku može kod sebe privući samo jednu kocku. Dizalica koja raznosi kocke ima 5 mogućih funkcija:

1) pomak u lijevu stranu
2) pomak u desnu stranu
3) pomak unaprijed
4) pomak unazad
5) promjenu stanja „uključenosti“ magneta

Prikaz funkcija prikazan je na *slici 2*.



Slika 2. Prikaz funkcija dizalice

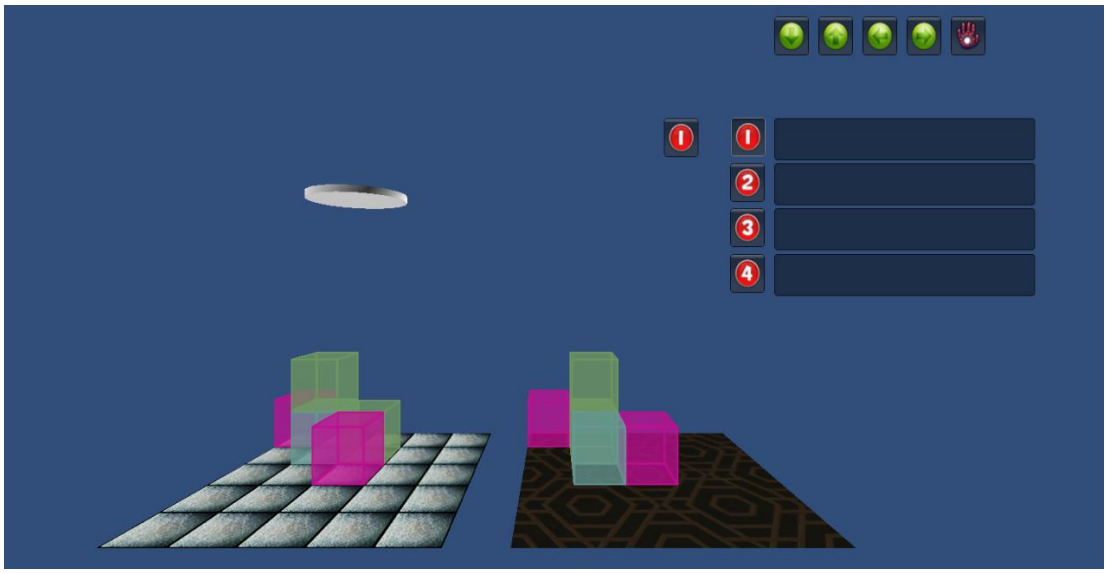
Dizalica može uključiti magnet samo kada se ispod nje nalazi kocka, a može ga isključiti samo ako je uključen. Time se u biti dobiva to da dizalica tom funkcijom „uzme“ kocku koja se nalazi ispod nje (ako takva postoji) ili ispusti kocku koju ima kod sebe. Kocke se razlikuju tako što su različitih boja. Moguće je postojanje više kocaka iste boje te se one tretiraju kao iste. Kocke iste boje se meeđusobno ne razlikuju, znači da ako na primjer postoje dvije kocke plave boje, nije bitno koja se nalazi na kojoj poziciji, bitno je da **neka** plava kocka bude na traženoj poziciji na kojoj treba biti **neka** nebitno koja plava kocka i tako za sve ostale.

3.2. Korisničko sučelje

Igra ima mogućnost izbora između postojećih zadanih problema. Problem se sastoji od početne konfiguracije kocaka te tražene konfiguracije

kocaka. Kada se izabere problem učitaju se početna i tražena konfiguracija te se prikažu na ekranu. U desnom gornjem kutu nalaze se ikone svih 5 funkcija dizalice. Ispod njih nalazi se prostor u koji se ikonice mogu odvući i tako graditi program. Postoje 4 trake za slaganje programa i svaka ima svoju ikonu. Trake se tretiraju kao potprogrami te se i same mogu odvlačiti u neki od potprograma. Prva traka (traka jedan) je glavni potprogram koji se „poziva“ i od kojeg kreće izvođenje kada korisnik klikne na gumb za pokretanje programa koji je stvorio.

Prikaz korisničkog sučelja nalazi se na *slici 3*.



Slika 3. Korisničko sučelje

Početna i tražena konfiguracija prikazani su kao dva panela, početna konfiguracija lijevo, tražena konfiguracija desno. Na svaki od panela moguće je kliknuti kako bi se on zumirao i lakše pregledavao.

3.3. Ideje za nadogradnju

Igra je trenutno funkcionalna no postoji mjesta za napredovanje. U ovom odjeljku bit će opisane funkcionalnosti koje su u planu za dodavanje.

3.3.1. Uvjetno grananje

U planu je dodavanje mogućnosti uvjetnog grananje. Uvjetno grananje koristilo bi se tako da se iznad instrukcije može postaviti boja. Kada krene izvođenje instrukcija koja ima iznad sebe ima definiranu boju, instrukcija će se izvesti samo u slučaju da dizalica kod sebe ima kocku te boje.

3.3.2. Povratna informacija (feedback)

Bilo bi dobra kada bi netko tko igra takvu igru znao nakon što riješi neki od problema kakvo je njegovo rješenje. Kvaliteta rješenja mogla bi se gledati prema broju korištenih instrukcija i duljini izvršavanja. Mogla bi se dodati i nekakva rang lista te sve skupa postaviti dostupno na Internetu, kako bi se igrači mogli međusobno uspoređivati.

3.3.3. Editor problema

Još jedna zgodna stvar bila bi omogućiti igračima da imaju mogućnost izrade vlastitih problema te njihovu pohranu. To bi također bilo zgodno povezati sa prethodnom idejom te igračeve probleme staviti negdje dostupne na Internetu kako bi ih i drugi mogli igrati.

Naravno za napredak ima mnogo mjesta, a ovo su samo neke od ranih ideja koje bi bilo zgodno realizirati.

4. Realizacija igre

Za realizaciju igre korišten je 3D programski pogon Unity te programski jezik C#. Unity u sebi već ima puno toga gotovog, ali neke stvari nisu baš najbolje realizirane. Poteškoće na koje se nailazilo prilikom izrade ove igre bit će opisane u ovom poglavlju kao i načini na koje su se realizirale funkcionalnosti igre.

4.1. Pohrana konfiguracije scene

Zadani problemi s kojima će se igraći ove igre suočiti su kao što je ranije opisano početna i tražena konfiguracija kocaka. Te konfiguracije spremljene su kao obične txt datoteke sljedećeg formata.

U prvom retku datoteke nalazi se broj kocaka početne konfiguracije. Nakon toga slijedi n redaka u kojima se nalaze podaci o pojedinoj kocki (n označava onaj prvi broj iz datoteke, broj kocaka početne konfiguracije). Redak koji sadrži podatke o kocki sadrži 4 cijela broja odvojena razmakom koji redom označavaju:

1) x koordinata početnog terena (lijevog)
2) y koordinata početnog terena (lijevog)
3) z koordinata početnog terena (lijevog)
4) indeks boje u polju boja

Dakle svaka kocaka ima svoju poziciju i boju. Svaka kocka je velika jednu jedinicu u prostoru igre po svim dimenzijama.

Nakon kocaka početne konfiguracije slijedi ponovo još jednom redci istog formata samo za kocke tražene konfiguracije. Broj kocaka početne i tražene konfiguracije treba biti isti. Program to eksplicitno ne provjerava, ali kako se korisniku trenutno neće dozvoliti da sam piše te datoteke program za sada pretpostavlja da je stvaratelj igre i njenih problema to napravio dobro (u

opisanom formatu sa zadanim pravilima). Primjer jedne takve datoteke prikazan je na u *ispisu 1*.

```
5
1 0 1 0
1 0 2 1
0 0 0 2
1 1 1 1
2 0 2 2
5
2 0 1 0
1 0 1 1
0 0 0 2
1 1 1 1
2 0 2 2
```

Ispis 1

4.2. Modeli i teksture

Grafički modeli korišteni za ovaj rad napravljenu su u samom Unityju iz razloga što su dosta jednostavni. Imamo model kocke koji je jedan od primarnih tijela. Kocka ima parametar *opacity* koji je podešen kako bi kocka bila prozirna. Također postoji i opcija mijenjanja boje, te se tako dinamički stvaraju kocke različitih boja.

Kao podloga koriste se dva panela. Kako bi bilo jednostavnije slagati kocke paneli su skalirani tako da jedan „*word unit*“ (jedinica za mjerenje udaljenosti u igri) bude velika onoliko koliko će biti i kocka. Tako se neko područje panela može definirati koordinatama kao da se radi o 2D polju programskih jezika. Na panele je zalijepljena jednostavna besplatna tekstura skinuta s Interneta.

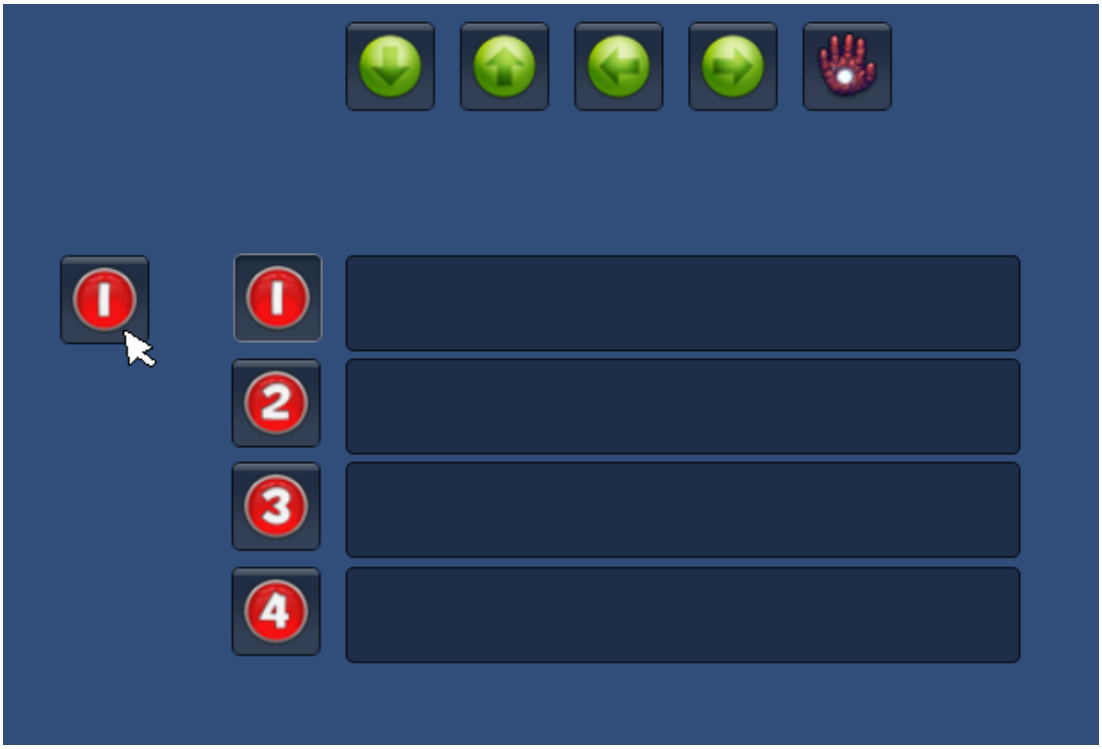
Zadnji model koji igra sadrži je dizalica. Dizalica je također „modelirana“ u Unityju, a napravljena je od običnog valjkastog tijela koje je spljošteno tako da dizalica liči na nekakav magnet čiju funkciju i obavlja.

4.3. Grafičko korisničko sučelje

Za izradu GUI-ja korištena je Unityjeva biblioteka za izradu GUI-ja. Unityjeva biblioteka za taj zadatak dosta je siromašna i ima par jednostavnih osnovnih stvari vezanih uz GUI. Za ovaj projekt nije bila potrebna prevelika funkcionalnost izvan tih okvira te su se potrebne stvari implementirale ručno bez korištenja tuđih biblioteka za izradu GUI-ja.

Uglavnom korištena komponenta bio je gumb (Button). Gumb se koristi kako bi se odabrao problem, kako bi se pokrenula igra, funkcije dizalice su realizirane gumbima koji se vuku, kako bi se pokrenuo stvoreni program, klikće se na gumb.

Kao što je ranije spomenuto za funkcije dizalice koriste se gumbi koji se mogu povući u traku programa. Ta traka je druga komponenta koja koristi Unityjev Box. Kako smo rekli da je Unityjeva biblioteka za izradu GUI-ja siromašna, funkcionalnost povlačenja gumba morala je biti dodatno implementirana.



Slika 4. Povlačenje gumba

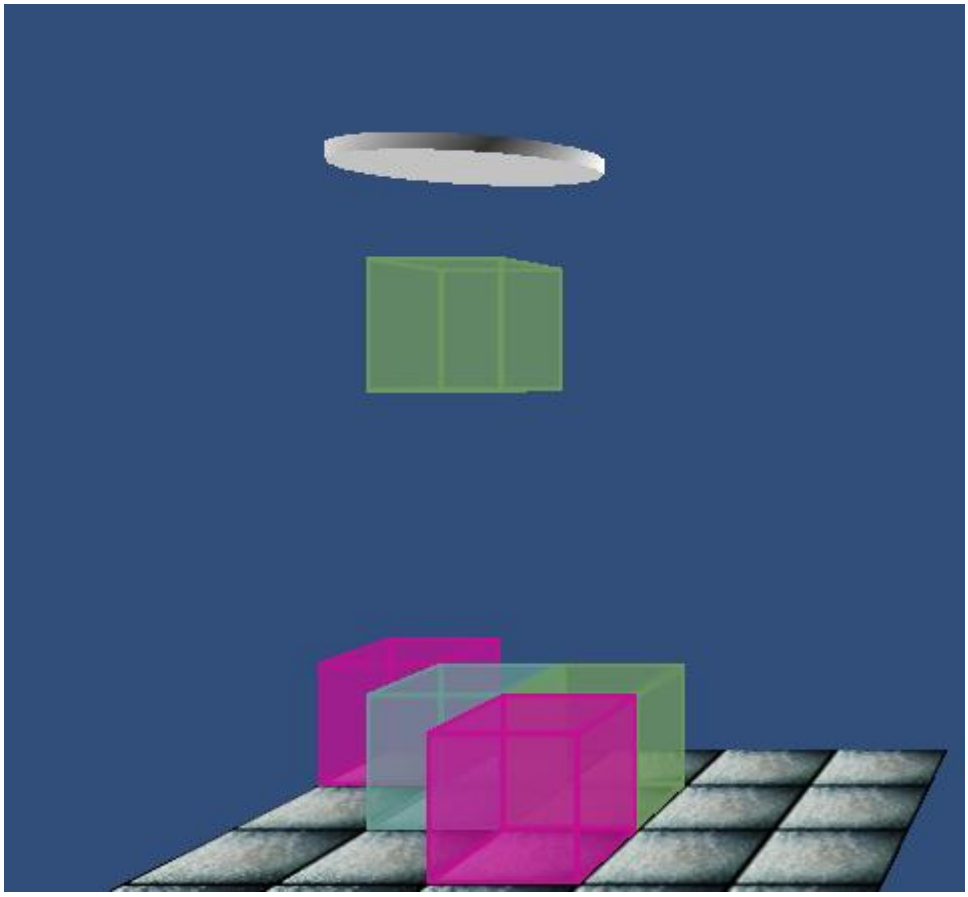
Na *slici 4.* prikazan je dio korisničkog sučelja s povlačenjem gumba. Povlačenje gumba realizirano je na način da se prilikom klika na gumb koji se može vući stvara novi isti takav gumb na istom mjestu gdje se nalazi miš te se postavlja jedna *boolean* varijabala koja predstavlja zastavicu da se gumb vuče. Dok god je ta zastavica postavljena na *true* i dok je pritisnut gumb miša novonastalom gumbu mijenja se lokacija na trenutnu lokaciju kursora miša. Kada se pusti gumb miša *boolean* varijabla se postavlja na *false* i gumb nestaje osim ako se nalazio iznad nekog od spremnika gdje se slaže program. U tom se slučaju gumb stvara i prikazuje iza zadnjeg gumba u tom spremniku.

4.4. Problemi pri realizaciji funkcionalnosti

Pri realizaciji funkcionalnosti dizalice bilo je dosta problema s povlačenjem i spuštanjem kocki jer Unityjeva ugrađena fizika ne radi kako bi

trebala. Unity ima nešto što se zove „rigidbody“. To je svojstvo koji se može dati bilo kojem objektu (npr. kocki) te to objektu daje svojstvo težine i gravitacije. Kako u Unityju ne može postojati više različitih gravitacijskih polja ili središta koja privlače predmete pokušaj realizacije povlačenja i spuštanja kocki napravljen je tako da se u trenutku povlačenja kocke toj kocki koja se povlači uključuje svojstvo *rigidbody* s negativnom gravitacijskom silom, a kada se kocka želi ispustiti daje joj se pozitivna gravitacija, te kad ona padne dolje, svojstvo *rigidbody* joj se isključi. No korištenjem ove ideje kocke su neobjašnjivo počinjale lebdjeti, te nakon mnogo sati traženja i ispravljanja grešaka i traženja rješenja na Internetu ustanovio sam da mnogo ljudi ima problema sa Unity fizikom i da savjetuju da se koriste vanjske biblioteke.

Kao konačno rješenje napravljena je klasa koja kocku pomiče prema gore i prema dolje. Za detekciju kocke koja se nalazi ispod dizalice i za početak nalazi li se uopće neka kocka ispod, korištena je tehnika bacanja zrake (eng. ray casting). Dizalica ispod sebe „baca“ zraku, računa presjecište pravca okomitog na dizalicu i onog što on prvo pogodi. Ako je to neka kocka i treba ju dignuti onda se ona podiže, ako je to podloga tada se ništa ne događa iz razloga što očito nema kocke koja bi se podigla. Kod spuštanja kocke bacanje zrake koristi se kako bi se izračunalo mjesto na koje se kocka treba staviti, to naravno neće uvijek biti razina 0 (na panelu) jer možemo slagati kocku na kocku.



Slika 5. Privlačenje kocke

Na *slici 5.* prikazano je privlačenje kocke prema dizalici.

4.5. Izvođenje programa

Izvođenje programa realizirano je dosta jednostavno. Svi gumbi odvučeni u spremnik za program spremljeni su te se po njima izvršavaju naredbe. Funkcije dizalice čija su izvođenja već implementirana ranije radi testiranja korištene su jedna po jedna ovisno o tome na kojem gumbu se trenutno nalazimo u izvođenja petlje. Dodana je provjera za slučaj da se dizalica pomakne izvan okvira panela, u tom slučaju program nije ispravan, te se ispisuje adekvatna poruka korisniku (igraču). Nakon izvršenja cijelog programa (svih instrukcija) provjerava se odgovara li pozicija svih kocaka na

lijevom panelu onima na desnom, ako odgovara program je ispravan, u suprotnom nije. Ispisuje se odgovarajuća poruka. Korisnik nakon izvršenja programa i daje ostaje na tom problemu te može mijenjati svoj program i ponovo ga pokretati.

5. Zaključak

Danas je programiranje sve više raširenо, te kod mladih skoro pa da postaje dio opće kulture. Stoga može biti dosta bitno djecu od mladih dana učiti načinu razmišljanja koje će im kasnije koristiti. Zgodno rješenje su razne igre koje su zarazne i zabavne osim što su edukativne. Takve igre osim razvijanja i pripremanja mozga za programiranje, općenito razvijaju igračev mozak te su s toga pogodne kao i bilo koje druge „logičke“ igre (mozgalice).

Igra razvijena u ovom projektu je pokušaj nečeg takvog. Igra je trenutno upotrebljiva i može se igrati, funkcionalnosti su implementirane, no smatram da je kako bi igra postala ugodna i privlačna, te ne dosadna potrebno još rada na detaljima.

Osim razvoja te igre u okviru ovog završnog rada bavio sam se i upoznavanjem Unity pogona. Unity je moderan i moćan pogon u kojem se vrlo lako mogu razviti 3D igre. Za fiziku i GUI preporučljivo je koristiti vanjske biblioteke. Prednost Unityja je što jednom napisan kod daje igru koja se može igrati na gotovo svim mobilnim platformama (Android, iOS, Windows Phone, BlackBerry), svim poznatim operacijskim sustavima (Linux, Windows, Mac OS), te igraćim konzolama i u web preglednicima. Posebno zanimljiva stvar je da se uz instaliran *Unity web player* igra može pokrenuti iz vanjskih repozitorija poput Dropboxa bez ikakve instalacije.

6. Literatura

[1] Vizualni programski jezici – pristup stranici 6.6.2013. http://en.wikipedia.org/wiki/Visual_programming_language

[2] Unity wiki stranice – pristup stranici od 15.3.2013. do 14.6.2013. http://wiki.unity3d.com/index.php/Main_Page

[3] Sličan projekt – pristupio stranici 28.5.2013. http://bennyzhang.com/?p=24

7. Sažetak

Ovaj rad opisuje što je to vizualno programiranje, te koje su mu sve primjene. Također opisuje izradu jednog alata (igre) za vizualno programiranje koji služi djeci za rano navikavanje na programski način razmišljanja i odraslima za zabavu i razvijanje vještina. Opisuju se neke funkcionalnosti Unityja (pogon koji se koristio za izradu igre), neke njegove mane i problemi na koje se nailazilo prilikom izrade igre. Za probleme na koje se naišlo također su opisana i rješenja tih problema koja su upotrijebljena u izradi igre.

Programiranje je danas veoma bitno jer se njime realizira veliki dio sve tehnologije koja se danas primjenjuje, te je s toga bitno djecu od mladih dana navikavati na takav način razmišljanja.

Unity je moderan i snažan alat za izradu 3D aplikacija (prvenstveno igara) koji podržava 3 programska jezika, a projekti jednom napravljeni u njemu dostupni su na gotovo svim platformama koje danas postoje, bilo web, stolne, mobilne ili igraće konzole.

Abstract

This paper describes what is visual programming and what are it's uses. Also it describes construction of such a tool (a game) for visual programming which is mainly designed for kids to learn how to think like programmers from early age. The paper describes some of Unity's functionality (Unity is the engine used for game construction), some of it's defects and problems found when game was in construction. For that problems there are described solutions made for this game.

Today programming is very important because with it big part of technology is realized, and that is why is important to learn children that kind of thinking from early age.

Unity is modern and powerful tool for construction of 3D applications (mainly used for games) which provides you with 3 programming languages, and project made ones can be published on almost all existing platforms today, web, desktop, mobile and game consoles.