Univerza
v Ljubljani
Fakulteta
za gradbeništvo
in geodezijo

Jamova 2
1000 Ljubljana, Slovenija
telefon (01) 47 68 500
faks (01) 42 50 681
fgg@fgg.uni-lj.si

Univerzitetni študij gradbeništva,
Konstrukcijska smer

Kandidat:
Ladislav Klinc

RAZVOJ ORODJA ZA IZDELAVO INTEGRIRANIH MODELOV V PROGRAMU SKETCHUP

Diplomska naloga št.: 3123

Mentor:
izr. prof. dr. Tatjana Isaković

Somentor:
viš. pred. dr. Tomo Cerovšek

Ljubljana, 2010

IZJAVA O AVTORSTVU

Podpisani **LADISLAV KLINC** izjavljam, da sem avtor diplomske naloge z naslovom:

"**Razvoj orodja za izdelavo integriranih modelov v programu Sketchup**".

Izjavljam, da se odpovedujem vsem materialnim pravicam iz dela za potrebe elektronske separatoteke FGG.

Ljubljana, 7.6.2010 Podpis:

BIBLIOGRAFSKO–DOKUMENTACIJSKA STRAN IN IZVLEČEK

UDK: **004.42:519.61/.64:69(043.2)**

Avtor: **Ladislav Klinc**

Mentor: **Izr. prof. dr. Tatjana Isaković**

Somentor: **Viš. pred. dr. Tomo Cerovšek**

Naslov: **Razvoj orodja za izdelavo integriranih modelov v programu Sketchup**

Obseg in oprema: **59 strani, 34 slik, 4 preglednice**

Ključne besede: **BIM, integrirani modeli, terminski plan, simulacije gradnje, 4D in 5D simulacije**

Izvleček

V diplomski nalogi je predstavljen razvoj dveh programskih orodij in sicer za izdelavo terminskih planov ter integriranih modelov v programu Sketchup, kjer je omogočena 4D in 5D simulacija gradnje. Program za izdelavo terminskih planov je izdelan kot spletna aplikacija na podlagi ogrodja Adobe Flex, pri zasnovi programa smo se osredotočili na izdelavo programa, ki bo enostaven za uporabo ter dostopen prek vsakega računalnika z internetno povezavo; program ima tudi funkcionalnost spremljanja sprememb ali zgodovine terminskega plana glede na posamezne aktivnosti. Dodana je tudi možnost vnosa in analize stroškov.

V drugem delu diplomske naloge smo v programu Sketchup omogočili izdelavo integriranih modelov in sicer s povezavo 3D elementov z aktivnostmi v terminskem planu. S tem smo dva programa povezali in omogočili simulacijo gradnje v programu Sketchup. Tudi tu je bil poudarek na izdelavi programa enostavnega za uporabo, ki bo razširil uporabo integriranih modelov in simulacij gradenj v gradbeništvu. Na koncu diplomske naloge je še uporabniški priročnik, ki bralca seznani z delovanjem in funkcionalnostmi obeh programov.

BIBLIOGRAPHIC–DOCUMENTALISTIC INFORMATION AND ABSTRACT

UDK: **004.42:519.61/.64:69(043.2)**

Author: **Ladislav Klinc**

Supervisor: **Assoc. prof. dr. Tatjana Isaković**

Cosupervisor: **Sen. lec. dr. Tomo Cerovšek**

Title: **Development of a software tool for creating integrated models in SketchUp**

Notes: **59 pages, 34 figures, 4 tables**

Keywords: **BIM, integrated models, building schedules, simulation of building process, 4D and 5D simulations**

Abstract

The present work describes the development of two software tools for creation of 4D and 5D simulations of the building process. One tool is used to create building schedules and is implemented as a web application using Adobe Flex framework. The main focus was to create a program that would be easy to use and available on any computer connected to the internet; this tool also enables a full audit trail of the operations used while creating a building schedule, further more it is possible to add financial data and their analysis.

The second tool creates integrated models within Sketchup and is made possible by the connecting 3D model with the building schedule. Thus we connected two seperate tools and created the possibility to create simulations of building process within Sketchup; here too the emphasis was on the ease of use and will hopefully broaden the use of integrated models and simulations of building process. The last part of the work is user's guide that describes how to operate both programs and shows their core functionalities.

ZAHVALA

Zahvaljujem se viš. pred. dr. Tomu Cerovšku ter izr. prof. dr. Tatjani Isaković za idejo diplomske naloge in za vso pomoč v času njenega nastajanja.

Zahvalil bi se tudi mami Slobodanki in očetu Tomažu, Demijanu, Goranu in Nini za vso podporo skozi leta študija.

Dodatna zahvala očetu za temeljit pregled in popravke diplomske naloge.

KAZALO VSEBINE

KAZALO SLIK

KAZALO PREGLEDNIC

1 UVOD

Pri projektiranju gradbenih objektov se v zadnjem času vse večkrat uporabljajo koncepti Informacijskega modela zgradbe (angl., *Building information model*) ali krajše BIM. To je pristop, s poudarkom na dodatku informacij modelu 3D. Pomembna prednost takšnega pristopa k projektiranju je, da vse sodelujoče stroke uporabljajo isti model objekta. Boljša komunikacija med udeleženci je ena izmed ključnih prednosti informacijskega modela zgradbe.

1.1 Problem

Glavni problem, ki ga diplomska naloga obravnava je sledeč: kako izdelati integrirani model stavbe s pomočjo katerega bo mogoče simulirati proces gradnje.

BIM omogoča povezovanje elementov 3D modela s terminskim planom gradnje, kar pomeni, da elementom dodatno pripišemo meta podatke kot so začetek in zaključek gradnje, odstotek opravljenega dela in podobno.

Pri zamisli diplomske naloge smo si ogledali rešitve za izdelavo integriranih modelov in simulacij gradnje. Te rešitve so v večini primerov vezane na programske pakete, ki so zelo obsežni in zaradi visoke cene marsikomu nedostopni. Zato smo želeli izdelati program za izdelavo integriranih modelov in simulacij gradnje, ki bi bil brezplačen in enostaven za uporabo.

1.2 Metodologija

Odločili smo se za izdelavo dveh med seboj ločenih, vendar povezanih programov.

Prvi bo namenjen izdelavi terminskih planov, katerega bo mogoče uporabljati na praktično vsakem računalniku z brskalnikom ter Adobe Flash Player-jem. Program bo dodatno omogočal uvoz terminskih planov izdelanih s programom Microsoft Project. Poleg tega bo mogoče definirati tudi

stroške izgradnje in s tem izdelavo 4D in 5D simulacije gradnje, s časom in stroškom kot dodatnima dimenzijama.

Drugi program bo namenjen integriranim modelom in simulaciji gradnje, z upoštevanjem terminskega plana objekta. Tu je bilo potrebno najprej izbrati primeren program za 3D modeliranje in odločili smo se za Google Sketchup. Program je zmogljiv, enostaven za uporabo, teče na operacijskih sistemih Windows ter Mac OS X in je v osnovni različici brezplačen. Sketchup ponuja tudi programski aplikacijski vmesnik–API (angl., *Application Programming Interface*) s katerim uporabniki po potrebi izdelajo dodatne programske funkcije.

Rezultat diplomske naloge sta delujoča programa, za izdelavo terminskih planov ter integriranih modelov in simulacij gradnje. Programa sta brezplačna ter enostavna za uporabo, uporabniki pa bodo lahko brez težav pri svojih projektih izdelali simulacijo gradnje, ter tako izkoristili prednosti integriranih modelov

1.3 Struktura naloge

Diplomska naloga vsebuje sedem poglavij, po uvodnem poglavju, je v drugem opisan informacijski model zgradbe—BIM, ki je pomemben napredek pri projektiranju v zadnjem času. Predstavljene so prednosti informacijskega modela v fazi načrtovanja, pred začetkom gradnje ter v fazi gradnje. Posebna pozornost je namenjena simulacijam gradnje z integriranimi modeli, saj je bila glavnina dela v sklopu diplomske naloge namenjena prav temu.

V tretjem poglavju je dan strnjen pregled orodij in programskih jezikov, ki smo jih pri izvedbi diplomske naloge potrebovali in bo bralcu omogočil lažje razumevanje snovi naslednjih poglavij.

V četrtem poglavju je opisana izdelava programa za terminske plane. Najprej sta opisana zasnova programa ter implementirane funkcionalnosti, nato so predstavljeni izvedba in način delovanja programa, ter delovanje nekaterih zanimivih in kompliciranih rešitev, ki smo jih uporabili.

Peto poglavje je namenjeno programu za simulacijo gradnje objektov. Prikazane so zahteve in diagram uporabe, zasnova ter potrebne funkcionalnosti. Na koncu je prikazana implementacija programa, ter dan opis nekaterih funkcionalnosti.

V šestem poglavju je prikazan uporabniški priročnik programov za izdelavo terminskih planov in integriranih modelov za simulacije gradnje

V zadnjem poglavju so podani zaključki diplomske naloge.

2 BIM

2.1 Uvod

Informacijski model zgradbe (angl., *Building information model*), ali krajše BIM, je v zadnjem času eden najbolj pomembnih napredkov na področju arhitekture, gradbeništva in inženiringa (v literaturi se pojavlja angleška kratica *AEC—Architecture, Engineering and Construction*). S tehnologijo BIM se izdela virtualni model objekta, ki ne obsega samo 3D model z natančno geometrijo pač pa tudi meta podatke, ki so potrebni za izvedbo, izdelavo in realizacijo objekta. Tako v BIM modelu pripišemo elementom iz 3D modela dodatne parametre kot so način izdelave, povezava s terminskim planom, cena in drugo. V projektu BIM so poleg arhitekture dodani tudi elementi kot so vodne in električne inštalacije, prezračevalne cevi in ostale prezračevalne komponente (angl., *HVAC—Heating, Ventilating and Air Conditioning*). BIM omogoča, da vsi sodelujoči pri projektu uporabljajo enak model objekta, dodatno pa olajša tudi sodelovanje in komunikacijo med različnimi panogami, in to ne samo med arhitekti, gradbeniki in inženirji, pač pa tudi z lastniki in upravitelji objekta ter drugimi skupinami.

2.2 Uporaba in prednosti BIM-a

BIM na mnoge načine izboljša načrtovanje, izdelavo in uporabo objekta. Kljub temu, da se gradbena industrija na BIM šele privaja, smo že priča velikim spremembam, glede na standardne 2D risbe ter uporabo in izmenjavo papirnih podlog. Oglejmo si prednosti BIM-a po fazah izdelave objekta.

2.2.1 Prednosti pred začetkom načrtovanja

Pred začetkom načrtovanja je omogočeno, da investitorji objekta preučijo smotrnost izgradnje objekta. Dobijo odgovor o okvirnem času izgradnje in ali je zamišljeni objekt sploh mogoče izdelati z razpoložljivimi finančnimi sredstvi. Investitor lahko izgubi veliko časa ter finančnih sredstev, če izdela detaljni načrt objekta, nato pa ugotovi, da bo objekt predrag ali pa bo gradnja predolga. Obstajajo možnosti, da se preko makro modelov in povezave modela z ustrezno bazo podatkov oceni strošek in trajanje gradnje.

Izdelava shematičnega modela objekta pred natančnim modelom omogoča investitorju tudi pregled ali objekt izponjuje pričakovanja glede funkcionalnosti in trajnosti. Začetno dobro načrtovanje omogoča izboljšanje v primeru, da se gradnja in načrtovanje nadaljujeta.

2.2.2 Prednosti v fazi načtovanja

Model BIM omogoča izboljšano vizualizacijo objekta v vseh fazah gradnje, to pa gre pripisati dejstvu da kot podlago uporabljamo 3D model, ne pa 2D risbe. Ker model pregledamo v računalniškem programu, lahko že v fazi načrtovanja odpravimo neskladnja v objektu, ki bi se sicer pojavile med gradnjo. Tako odpravljanje napak je veliko cenejše od napak, ki jih moramo popraviti na gradbišču. Za odpravljanje neskladij je izredno pomembno sodelovanje sodelujočih panog. Uporaba 3D modela omogoča tudi enostavno generiranje tlorisov ter prerezov, kjer smo lahko prepričani, da bodo risbe izdelane v skladu s 3D modelom.

Ker pri BIM vsakemu elementu pripišemo dodatne parametre, lahko natančno in hitro izračunamo količino elementov ter ploščine prostorov, kar omogoča zanesljivejšo oceno stroškov izgradnje.

Natatančen popis vseh elementov, količin in ploščin omogoča tudi bolj natančno snovanje terminskega plana. Poznavanje količin v povezavi z bazo podatkov o produktivnosti dela omogoča

določevanje trajanja izgradnje vsakega elementa v objektu. Iz baze podatkov lahko za vsako delo izračunamo število delavcev, ki so potrebni, da se določeno delo opravi v določenem času.

Na koncu omenimo še možnosti izboljšanja energijske bilance ter trajnosti objekta. Povezava modela z orodjem za energetsko analizo, nam omogoča, da objekt med fazo načrtovanja optimiziramo tako, da bo objekt čimbolj energijsko neodvisen, da bo izkoriščal naravne vplive, kot je sončno sevanje ter da bosta izgradnja in upravljanje objekta čim manj negativno vpivala na okolje.

2.2.3 Prednosti v fazi gradnje

Načrtovanje objekta z integriranim modelom zahteva povezovanje 3D elementov z aktivnostmi v terminskem planu. S povezovanjem je za vsak element v virtualnem objektu točno določen začetek in konec gradnje. To omogoča prikaz simulacije gradnje za celotni objekt, kjer vidimo kakšen bo potek gradnja iz dneva v dan. Več o integriranih modelih in simulacijah gradnje bo govora v naslednjem poglavju.

Elemente iz 3D modela lahko uporabimo tudi za izdelavo prefabriciranih elementov, kar zopet skrajša čas izdelave natančnih risb, saj dobimo veliko podatkov kar iz BIM modela.

Ker razpolagamo z veliko količino informacij o objektu in njegovih sestavnih delih, lahko gradnjo optimiziramo. Sodelovanje izjavalca objekta in podizvajalcev omogoča, da se delo opravi, takrat, ko je za podizvajalčevo delo vse pripravljeno. To sodelovanje je odločilno če želimo zmanjšati časovne izgube na gradbišču ter vpeljati *Just-in-time delivery* oziroma prihod elementov ter delavcev takrat, ko jih na gradbišču potrebujemo. S tem je omogočeno zmanjšanje skladiščenja elementov in materiala, pred vgradnjo.

2.3 Simulacija gradnje s 4D in 5D modeli

Simulacijo gradnje omogoča povezava 3D modela s terminskim planom, ker imamo poleg treh prostorskih dimenzij, še četrto dimenzijo in sicer čas, taki simulaciji pravimo 4D simulacija. Ko dodamo še eno dimenzijo in sicer strošek gradnje, pridemo do 5D simulacije, ki hkrati spremlja časovne spremembe objekta ali njegovih elementov ter stroške.

Simulacija gradnje služi predvsem kot komunikacijsko orodje za odkirvanje mogočih zastojev na gradbišču in kot metoda za izboljšanje sodelovanja med delavci na gradbišču. Prednost simulacije gradnje je tudi vizualizacija poteka gradnje predno se ta dejansko začne.

2.3.1 Prednosti simulacije gradnje

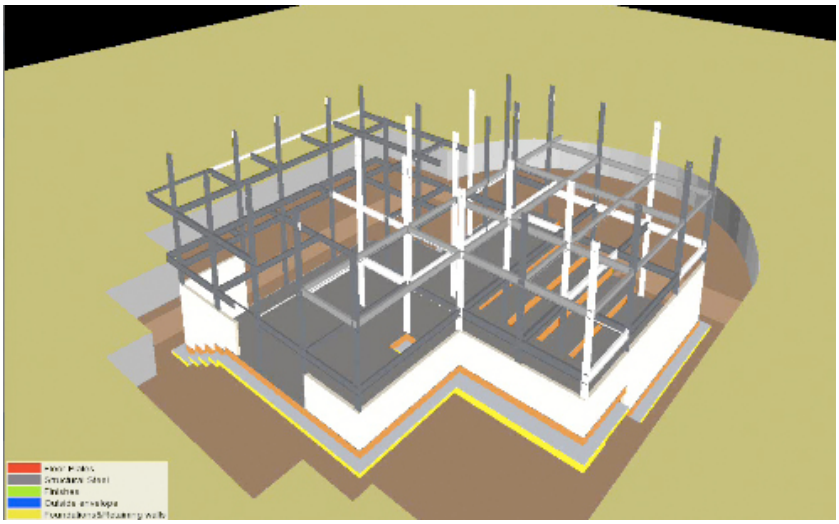
Poglavite prednosti uporabe simulacije gradnje so:

- ***Komunikacija:*** izdelovalci terminskega plana lahko s pomočjo simulacije gradnje lažje vizualno predstavijo idejo o gradnji objekta. Ker gre za vizualizacijo terminskega plana na 3D modelu, si je lažje predstaviti potek gradnje, kot z Ganttovim diagramom.
- ***Predlogi širše javnosti:*** simulacija gradnje se velikokrat uporabi za predstavitev gradnje objekta širši javnosti, mogoč pa je tudi prikaz vpliva gradnje na promet.
- ***Logisika gradbišča:*** izdelovalci terminskih planov lahko s pomočjo integriranega modela in simulacijo gradnje načrtujejo skladiščenje materiala, dostop do gradbišča ter lokacijo velikih strojev, prikolic in podobno.
- ***Primerjava različnih terminskih planov ter beleženje napredka gradnje:*** projektni manedžerji lahko primerjajo izdelavo objekta po različnih terminskih planih, pravtako lahko ugotovijo ali poteka gradnja v skladu s terminskim planom ali pa je prišlo do zastojev.

2.3.2 Orodja za simulacijo gradnje

Obstaja več orodij, ki z BIM tehnologijo omogočajo izdelavo simulacije gradnje. Taka orodja in njihovi proizvajalci so:

- Virtual Construction 5D—VICO Software
- ProjectWise Navigator—Bentley
- Project 4D ConstructSim—CommonPoint
- Visual Simulation—Innavaya
- JestStream Timeliner—Navisworks
- Synchro 4D—Synchro Ltd.



Slika 1: Simulacija gradnje z Virtual Construction 5D

3 PREGLED ORODIJ IN PROGRAMSKIH JEZIKOV ZA RAZVOJ

3.1 Sketchup

Sketchup je programsko orodje za 3D modeliranje, namenjeno arhitekteom, gredbenikom, piscem računalniških iger, skratka vsem, ki morajo z računalniškim programom preprosto in učinkovito izdelati 3D modele.

Sketchup je leta 2000, razvilo podjetje @Last Software. Del tega programa omogoča uporabnikom pisanje vtičnikov (angl., *plugins*), ki razširijo možnosti uporabe programa Sketchup, kot tudi njegovih orodij; primer tega je zelo uporabna razširitev, ki omogoča v Sketchup-u narejen model uvoziti v Google Earth.

Google je leta 2006 kupil podjetje @Last Software, z razvojem programa pa so nadaljevali in v letu 2008 izdali že njegovo sedmo različico.

Sketchup se od večine drugih programov za 3D modeliranje razlikuje po tem, da je izjemno enostaven za uporabo, kljub temu pa lahko z njim izdelamo izjemno natančne 3D modele, ki so primerjivi z drugimi tovrstnimi programi. Ideja programa sloni na tem, da bi bilo modeliranje podobno risanju na papir (od tod tudi ime programa, saj sketch, v angleščini pomeni risba). Če želimo v programu narisati premico, to naredimo kar s premikom miške, premico lahko hitro pretvorimo v ploskev, katero v 3D objekt spremenimo z vlečenjem ali rinjenjem, metodo ki jo ima Sketchup patentirano (angl., *push and pull*).

Sketchup izhaja v brezplačni in plačljivi profesionalni različici. Brezplačna različica omogoča praktično vse kar običajni uporabnik potrebuje, profesionalna različica pa vsebuje še program Lay-Out za integracijo 2D in 3D modelov in več opcij za izvoz 3D modela v .pdf, .eps, .dwg in druge formate.

3.1.1 Ruby

Ruby je dinamičen, odprtokoden, objektno usmerjen programski jezik, ki ga je v letu 1995 izdelal Japonski programer Yukihiro Matsumoto. Navdih za razvoj jezika je črpal iz programskih jezikov kot so Perl, Smalltalk, Eiffel, Ada in Lisp. Avtor je povedal, da je želel izdelati jezik bolj zmogljiv kot Perl in bolj objektno usmerjen kot Python.

Posebnost in značilnost Ruby-ja je v tem, da je v Ruby-ju vse objekt. Kot primer omenimo, da je `Number` v večini programskih jezikov primitiven podatkovni tip, v Ruby-ju pa objekt, preko katerega izvajamo poljubne metode. Na Ruby-jevi spletni strani je prikazan naslednji primer:

```
5.times { print "We *love* Ruby -- it's outrageous!" }
```

Tu na številu 5 izvedemo metodo `times`, ki bo pet krat ponovila vse, kar je zapisano znotraj zavihanih oklepajev. V Ruby-ju lahko vsakemu bitu informacij dodelimo atribute in na njih izvajamo metode. Tudi vrednosti kot so `true`, `false` in `nil` so objekti. Oglejmo si nekaj primerov kode:

Uvedemo dve spremenljivki `x` in `y`, `x` dodelimo vrednosti 5 in `y` vrednost 8.

```
x, y = 5, 8
```

Izračunamo x^3 (enako kot bi zapisali `x*x*x`) z Ruby-jevo operacijo ** potenciranja.

```
x**3
```

V spremenljivko `z` zapišemo ostanek pri deljenju spremenljivke y in števila 3, to je 2.

```
z = y % 3
```

V spremenljivki `ploscina` je ploščina kroga z radijem 2.3, pri tem iz modula Math uporabimo konstanto π. Dostop do konstant znotraj modulov izvaja dvojno dvopičje, (::), dostop do metod znotraj modula pa pika (.).

```
ploscina = Math::PI * (2.3 ** 2)
```

Oglejmo si še primer pogojnega `if else` stavka:

```
if y >= 0
  x = y
else
  x = -y;
end
```

V Ruby-ju je mogoče vsak problem rešiti na več načinov, tako enako funkcionalnost kot v predhodnjem primeru dosežemo tudi z:

```
x = y >= 0 ? y : -y
```

Funkcije v Ruby-ju definiramo z `def`, konec metode označimo z `end`, vse kar je znotraj bloka `def` ter `end`, se izvede pri klicu funkcije. Metode so funkcije, ki se izvedejo na objektu.

```
def diagonala(a, b)
  Math.sqrt(a**2 + b**2)
end
```

Funkcija `diagonala` potrebuje dva argumenta, dolžini katete a in b, izračuna pa dolžino hipotenuze. Ruby ne potrebuje ukaza `return` če zelimo, da nam funkcija vrne vrednost, saj Ruby vedno vrača vrednost zadnje izvršenega ukaza.

Za konec si oglejmo še kako v Ruby-ju izdelamo objekt; objekt se definira s `class`, kot funkcija pa se konča z `end`. Spremenljivke objekta (angl., *instance variables*) se od ostalih spremenljivk razlikujejo po tem, da se začnejo z znakom @. Spremenljivka objekta je lahko definirana le znotraj metode, če pa želimo spremenljivko definirati izven metod objekta uporabimo razredno spremenljivko (angl., *class variable*), katere ime se začne z @@.

```
class Krog
  def polmer(r)
          @polmer = r
  end
  def get_polmer
          @polmer
  end
  def ploscina
    Math::PI * @polmer**2
  end
  def obseg
          2 * Math::PI * @polmer
  end
end

#uporaba objekta krog
krog = Krog.new
krog.polmer(3.22)     #nastavi polmer na 3.22
puts krog.ploscina    #izpise 32.5732892694804
puts krog.obseg       #izpise 20.2318566891183
```

3.1.2 Ruby Sketchup API

Sketchup Ruby API, je knjižnica Ruby metod in razredov, ki omogočajo izdelavo novih funkcionalnosti programa Sketchup. Programom, izdelanim z Ruby API-jem, pravimo vtičniki.

Prek Ruby kode komuniciramo s Sketchupom na dva načina:

1. s pisanjem kode, kot skripta, katero shranimo v Plugins direktorij v Sketchup-u in se navadno požene prek menija Plugins
2. s pisanjem kode direktno v Sketchup-ovo konzolo (Window—Ruby Console)

Ruby API sestavlja prek 80 razredov (angl., *class*), kot so `Entity, Model, Menu, Selection, Color, Texture, Toolbar, UI, WebDialog`.

Kot primer Ruby API kode si oglejmo sledeči primer, kjer iz aktivnega modela dobimo podatke o vseh elementih ter uporabniku s pogovornim oknom prikažemo število 3D elementov po tipu, kot so `edge` (rob), `face` (ploskev), `image` (slika), `componentInstance` (komponenta).

```
def sort_count_elements
  model = Sketchup.active_model
  ent = model.entities
  edge=face=constructionLine=componentInstance=constructionPoint=image=0
  ent.each {|x|
   edge += 1 if x.is_a? Sketchup::Edge
   face += 1 if x.is_a? Sketchup::Face
   constructionLine += 1 if x.is_a? Sketchup::ConstructionLine
   componentInstance += 1 if x.is_a? Sketchup::ComponentInstance
   constructionPoint += 1 if x.is_a? Sketchup::ConstructionPoint
   image += 1 if x.is_a? Sketchup::Image
  }

  UI.messagebox "Nr edges: #{edge}\n
  Nr Faces: #{face}\n
  Nr constructionLine: #{constructionLine}\n
  Nr componentInstance: #{componentInstance}\n
  Nr constructionPoint: #{constructionPoint}\n
  Nr images: #{image}"
end
```

Najprej definiramo funkcijo z imenom `sort_count_elements`, zatem v spremenljivko `model` shranimo aktivni model, za kar uporabimo `active_model` metodo razreda `Sketchup`. Nadalje v spremenljivko `ent` shranimo vse elemente modela. Ker Sketchup vrne elemente v obliki polja (angl., *array*), bo tudi `ent` polje.

Spremenljike, s katerimi bomo šteli elemente določenega tipa v modelu, nastavimo na 0. Z metodo `ent.each`, lahko pregledamo vse elemente (v našem primeru so to elementi iz 3D modela), ki se nahajajo v polju `ent`. V kolikor je element znotraj polja `ent` enak `Sketchup::Edge`, bomo spremenljivko `edge` povečali za 1 in enako storimo tudi za ostale tipe 3D elementov.

Z zadnjim ukazom prek `UI.messagebox` izpišemo število elementov različnih tipov, ki se nahajajo v modelu.

3.2 Flex in ActionScript 3.0

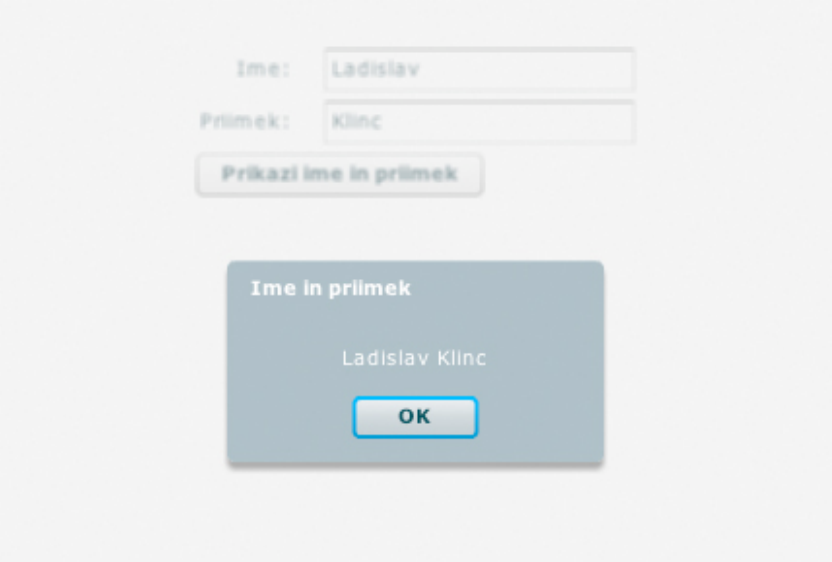
Flex je programski paket, podjetja Adobe Systems za izdelavo interaktivnih spletnih aplikacij, ki temeljijo na Flash platformi. Prav dejstvo, da je Flex baziran na Flash platformi je eden izmed njegovih najboljših karakteriskik. Flash se v računalništvu uporablja že vrsto let. Flash Player je vtičnik za brskalnik, ki poganja Flash datoteke in je na voljo na 99.1% vseh računalnikov, ki so se sposobni povezati na svetovni splet. Tako visok odstotek dosega nima niti Microsoft Windows, niti Java podjetja Sun. Razširjenost Flash-a pomeni, da je aplikacija izdelna v Flex-u na voljo skoraj vsakemu uporabniku svetovnega spleta, ne glede na operacijski sistem. Velika prednost Flash-a je, neodvisnost od uporabljenega brskalnika. Pri izdelavi internetnih aplikacij se velikokrat pojavi nemalo težav z drugimi orodji (recimo z AJAX-om), prav zaradi nekompatibilnosti brskalnikov.

3.2.1 Označevalni jezik MXML

Flex uporablja označevalni jezik, (angl., *markup language*), MXML za izdelavo in nadziranje gradnikov, kot tudi za sam izgled aplikacije. Prva vrstica v Flex aplikaciji je enaka prvi vrstici vsakega XML dokumenta, ki deklarira verzijo XML-ja ter kodiranje dokumenta. Na drugi vrstici se pojavi oznaka mx:Application, znotraj katere se nahaja celotna koda programa.

Oglejmo si aplikacijo izdelano z Flex-om, ki nas najprej vpraša za ime in priimek in ga nato izpiše v izkočnem oknu.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">
<mx:Script>
        <![CDATA[
        import mx.controls.Alert;
                private function prikaziIme():void{
                        var imeInPriimek:String = ime.text + ' ' + priimek.text;
                        Alert.show(imeInPriimek, 'Ime in priimek');
                }
        ]]>
</mx:Script>
<mx:Form verticalGap="5">
        <mx:FormItem label="Ime:">
                <mx:TextInput id="ime"/>
        </mx:FormItem>
        <mx:FormItem label="Priimek:">
                <mx:TextInput id="priimek"/>
        </mx:FormItem>
        <mx:Button label="Prikazi ime in priimek" click="prikaziIme()"/>
</mx:Form>
</mx:Application>
```



Slika 2: Prikaz enostavne aplikacije v Flex-u

3.2.2 ActionScript 3.0

Za interaktivnost v Flex aplikaciji skrbi moderni programski jezik Actionscript 3.0, napisan po specifikacijah četrte izdaje ECMAScripta. ActionScript 3.0 je objektno usmerjen programski jezik za izdelavo aplikacij in skript, ki se izvajajo v Flash okolju. Po sintaksi spominja na Javo in C#.

Kot primer ActionScript 3.0 kode si oglejmo, deklaracijo spremenljivke `sirina`, ki je tipa `int`– naravno število (angl., *integer*), spremenljivki pa damo vrednost 36:

```
var sirina:int = 36;
```

Z naslednjo kodo ustvarimo `for` zanko, ki se izvede 10 krat:

```
for (var i:int = 1, i <= 10; i++){
  //koda znotraj zanke se bo izvedla 10 krat
}
```

Na koncu si oglejmo še kako v ActionScriptu 3.0 ustvarimo objekt:

```
public class Krog {
        var _polmer:Number;
        public function Krog(polmer:Number) {
                _polmer = polmer;
        }
        public function obseg():Number {
                return 2*Math.PI*_polmer;
        }
}
```

Tako lahko ustvarimo objekt `Krog`, za katerega podamo polmer in izračunamo njegov obseg, rezultat pa zapišemo v konzoli:

```
var krog:Krog = new Krog(10);
trace(krog.obseg());
```

3.2.3 Flex Builder

Flex platforma je brezplačna, zato lahko aplikacije ustvarimo v poljubnem urejevalniku besedila, prevod aplikacije v strojno kodo pa izvede Flex prevajalnik. Flex Builder je produkt, ki ga je tako kot Flex izdalo podjetje Adobe Systems. To je razvojno okolje (angl., *Integrated Development Environment–IDE*), ki je zasnovano na odprtokodnem razvojnem okolju Eclipse IDE.

Program Flex Builder vsebuje:

- orodja za lažje pisanje programske kode
- možnost vizualne izdelave uporabniškega vmesnika
- interaktivno vizualizacijo podatkov (povezovanje gradnikov in podatkov brez programske kode)
- urejanje izgleda aplikacije z uporabo CSS-a (akronim za *Cascading Style Sheets*) in grafičnim spreminjanjem lastnosti gradnikov
- dobra povezanost z ostalimi produkti podjetja Adobe Systems kot so Photoshop, Illustrator, Flash in Fireworks
- zmogljiva orodja za testiranje in optimizacijo aplikacije

Uporaba Flex Builder-ja je priporočljiva za razvoj Flex aplikacij, saj ponuja vrsto orodij, ki razvijalcu olajšajo delo. Ena najboljših lastnosti programa je, da nam pri pisanju kode pomaga z namigi o metodah in funkcijah, ki so na voljo pri nekem objektu ali spremenljivki.

Program ima tudi zmogljiv razhroščevalnik, ki aplikacijo v izbranem trenutku zaustavi in prikaže njeno stanje: podatke o spremenljivkah in njihovih trenutnih vrednostih. Uporaba razhroščevalnika omogoča uspešno in hitro odpravljanje napak.

4 PROGRAM ZA IZDELAVO TERMINSKIH PLANOV

Izdelava programa za enostavne terminske plane je bila prva naloga pri snovanju diplomske naloge. Želeli smo ustvariti orodje, ki bi bilo uporabno kar največjemu številu uporabnikov na različnih operacijskih sistemih. Odločili smo se za izdelavo tako imenovane spletne aplikacije, saj bo celotna aplikacija delovala znotraj brskalnika. Nadalje smo se odločili za uporabo ogrodja (angl., *framework*) Flex, ki ga je na osnovi tehnologije Flash izdalo podjetje Adobe Systems.

Največja prednost uporabe ogrodja Flex je, da lahko aplikacijo izdelamo tako, da bo delovala v vseh brskalnikih, ki imajo nastavljen dodatek Adobe Flash 10, ne bo pa nam potrebno paziti na različno obnašanje brskalnikov kot so Internet Explorer 6 in 7, Firefox, Chorme, Opera. Flash aplikacija teče namreč znotraj brskalnika v Flash Player-ju, ki deluje enako v vseh brskalnikih.

4.1 Zahteve

V programu za izdelavo terminskih planov smo se osredotočili na naslednje zahteve:

Funkcionalne zahteve

- povezovanje aktivnosti
  - SS — povezava začetek–začetek, angl., *start–start*;
  - FF — povezava zaključek–zaključek, angl., *finish–finish*;
  - FS — povezava zaključek–začetek, angl., *finish–start*;
- določanje zamika pri povezavah (angl., *lag*);
- možnost vnosa večih povezav za vsako aktivnost;
- preprečevanje neveljavnih povezav;
- določanje datuma začetka in zaključka aktivnosti;
- določanje in spreminjanje hierarhije aktivnosti;

- vpis stroškov;
- zapis in prikaz zgodovine upravljanja terminskega plana;
- prikaz zgodovine izbrane aktivnosti;
- analiza vpisanih stroškov;
- omogočiti Undo–Redo (“razdri zadnje–razdrto povrni”) funkcionalnost.

Tehnične zahteve

- dostopnost–spletna aplikacija
- enostavna uporaba
- izvoz terminskega plana v formatu XML;
- uvoz terminskega plana iz programa MS Project;

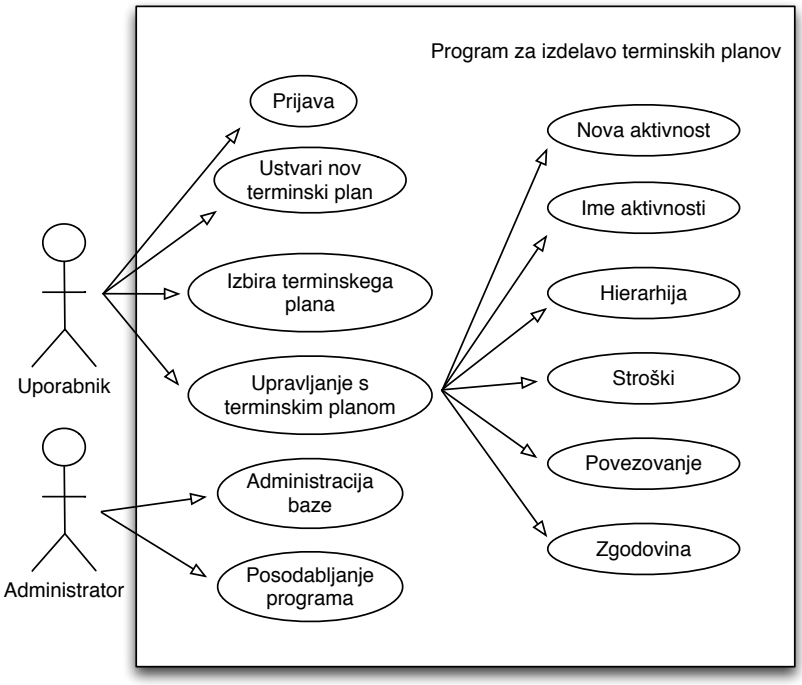
4.2 Diagram uporabe

Pred začetkom implementacije programa za izdelavo terminskih planov, smo izdelali diagram uporabe na sliki 3. Na diagramu so prikazane možnosti, ki jih bo imel uporabnik ter administrator programa. Diagram uporabe, nam kasneje služi kot vodilo pri implementaciji programa.

4.3 Zasnova

Program za izdelavo terminskih planov smo se odločili izdelati, ker nismo našli takega, ki bi zadostil našim zahtevam ter hkrati bil izdelan kot spletna aplikacija. Za izdelavo terminskih planov obstajajo programi, ki vsebujejo več funkcionalnosti, kot naša aplikacija, vendar cilj ni bil izdelati program, ki bi bil zmogljivejši od tistih na trgu, pač pa takega, ki bo:

- dostopen vsakomur preko na interneta,
- brezplačen,
- enostaven za uporabo.



Slika 3: Diagram uporabe programa za izdelavo terminskih planov

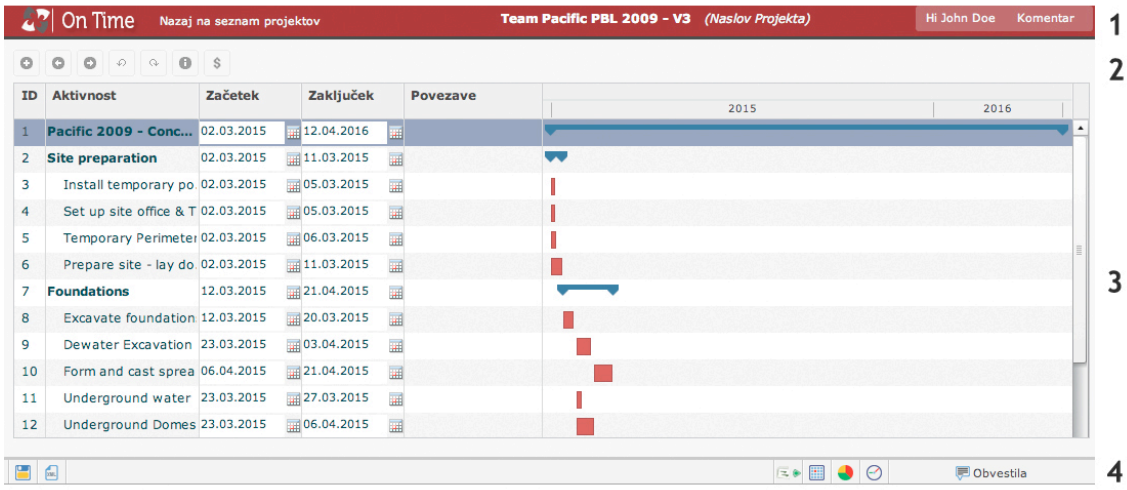
Spletna aplikacija ima nekaj prednosti pred običajnimi programi: ni jih potrebno namestiti na posameznem računalniku. Ker spletna aplikacija deluje znotraj brskalnika jo je veliko lažje izdelati tako, da deluje skoraj na vseh računalniških sistemih v uporabi danes.

Dodatna prednost spletne aplikacije je lažja distribucija ter posodobitev. Program je postavljen na oddaljenem strežniku, uporabnik do programa dostopa prek brskalnika in vedno uporablja najnovejšo verzijo programa.

Največja prednost izdelave programa kot spletne aplikacije pa je, da se lažje prilagodi za uporabo znotraj programa Sketchup, ki omogoča, da za uporabniški vmesnik (angl., *user interface–UI*) uporabimo spletno aplikacijo.

4.3.1 Uporabniški vmesnik

Uporabniški vmesnik je razdeljen na 4 enote (slika 1). V vrhnji vrstici označen z (1), je ime programa, gumb za navigacijo po projektih, ime odprtega projekta, na desnem delu vrstice okno z imenom uporabnika, ter možnostjo posredovanja komentarja avtorjem programa. V vrstici (2) so orodja za upravljanje s terminskim planom. Pretežni del zaslona je namenjen terminskemu planu, na sliki označen s (3). V vrstici na dnu (4) so orodja (z leve proti desni): shranjevanje projekta, izvoz v formatu XML, uvoz terminskega plana iz MS Project-a, nastavitve izgleda terminskega plana, grafični prikaz stroškov, pregled zgodovine in obvestil.



Slika 4: Pregled uporabniškega vmesnika

Terminski plan ponuja možnost prikazovanja naslednjih podatkov:

- ID—zaporedna številka aktivnosti
- GUID—unikatna identifikacijska številka
- Aktivnost—ime aktivnosti
- WBS—zaporedna številka glede na hierarhijo
- Trajanje—označuje dolžino trajanja aktivnosti v dnevih

- %—odstotek opravljenega dela
- Začetek—datum začetka aktivnosti
- Zaključek—datum zaključka aktivnosti
- Povezave—navaja vse povezave z drugimi aktivnostmi
- Ganttov diagram
- Stroški aktivnosti

Zaradi velikega števila stolpcev, ki nastopajo v terminskem planu, lahko uporabnik določi kateri podatki bodo vidni. S terminskega plana ni mogoče skriti le stolpcev zaporedne številke, imena aktivnosti ter Ganttovega diagrama. Pri zagonu programa, se odvisno od velikosti zaslona prikaže toliko stolpcev v terminskem planu, da je prikaz mogoč brez horizontalnega drsnika (minimalna širina za prikaz brez drsnika je 800 pik).

4.3.2 Povezovanje aktivnosti

Povezovanje aktivnosti je ena izmed temeljnih nalog programa za izdelavo terminskih planov. Cilj povezovanja je, da se aktivnosti v terminskem planu izvedejo v določenem vrstnem redu. Obstajajo štiri vrste povezav med aktivnostmi:

1. ***FS zaključek–začetek*** (angl., *finish–start*), datum začetka aktivnosti bo enak datumu zaključka povezane aktivnosti. V programu je ta vrsta povezave privzeta,
2. ***FF zaključek–zaključek*** (angl., *finish–finish*), datum zaključka aktivnosti bo enak datumu zaključka povezane aktivnosti,
3. ***SS začetek–začetek*** (angl., *start–start*), datum začetka aktivnosti bo enak datumu začetka povezane aktivnosti,
4. ***SF začetek–zaključek*** (angl., *finish–start*), datum zaključka aktivnosti bo enak datumu začetka povezane aktivnosti, povezava se redko uporablja in zato v programu ni implementirana.

V programu, smo implementirali povezovanje ene aktivnosti z več drugimi aktivnostmi, kjer program izbere za kritično povezavo tisto, ki ima v terminskem planu za dano aktivnost najkasnejši datum zaključka.

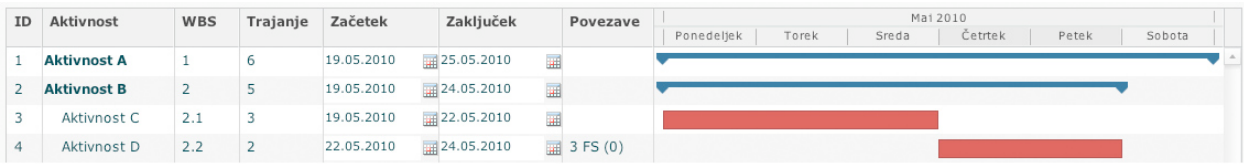
Poleg povezovanja aktivnosti na tri načine, (FS, FF in SS), program omogoča tudi določanje zamika (angl., *lag*) vsake povezave, kar pomeni, da se začetek aktivnosti glede na povezavo prestavi za toliko dni, kolikor je zamik.

Veliko pozornosti pri implementaciji povezovanja aktivnosti je bilo posvečeno preprečevanju neveljavnih povezav med aktivnostmi. Pregled povezav je podrobneje opisan v poglavju 4.4.2.

4.3.3 Zgodovina aktivnosti

Pri mnogih programih za izdelavo terminskih planov smo opazili, da je skoraj nemogoče spremljati potek izdelave terminskega plana od začetka do zaključka. Zato smo se odločili, da si bomo v našem programu zapisovali celotno zgodovino vseh opravil, ki se izvršijo. Spremembe, ki jih beležimo so lahko ročne ali avtomatične. Ročno spremembo opravi uporabnik, avtomatične spremembe po potrebi opravi računalniški program.

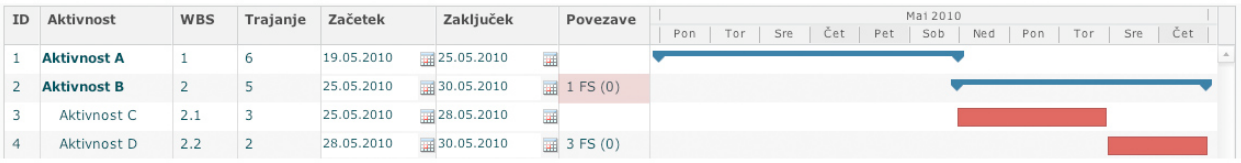
Za pojasnilo razlike med ročnimi ter avtomatičnimi spremembami si oglejmo sledeči primer. Imamo štiri aktivnosti A, B, C ter D. Aktivnosti C in D sta hierahično podrejeni aktivnosti B; Aktivnosti D predpišemo povezavo tipa FS z aktivnostjo C. Izgled terminskega plana je na sliki 5.



Slika 5: Prikaz terminskega plana po predpisani povezavi

Sedaj aktivnosti B določimo povezavo tipa FS z aktivnostjo A; Ta povezava je zapisana v zgodovini kot ročna. Dodatna povezava povzroči spremembo datuma začetka aktivnosti B, C ter D, saj smo

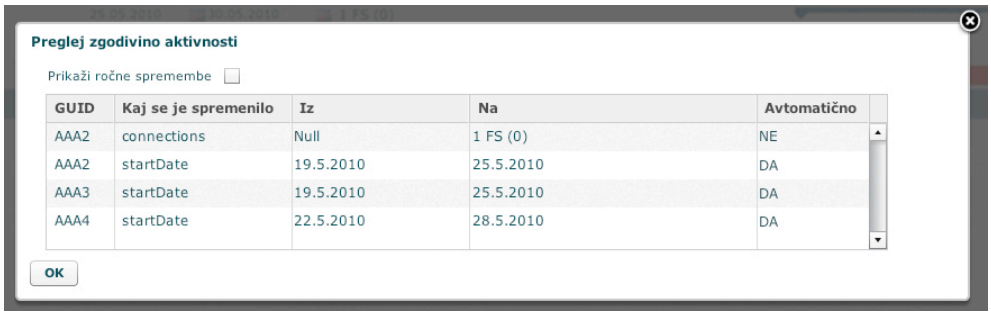
določili, da se bo B začela takrat, ko bo A zaključena. Te spremebe opravi program in jih v zgodovino vpiše kot avtomatične. Spremembo terminskega plana prikazuje slika 6, zgodovino sprememb pa slika 7.



ID	Aktivnost	WBS	Trajanje	Začetek	Zaključek	Povezave
1	**Aktivnost A**	1	6	19.05.2010	25.05.2010	
2	**Aktivnost B**	2	5	25.05.2010	30.05.2010	1 FS (0)
3	Aktivnost C	2.1	3	25.05.2010	28.05.2010	
4	Aktivnost D	2.2	2	28.05.2010	30.05.2010	3 FS (0)

Slika 6: Končni izgled terminskega plana

Ker vsako spremembo v terminskem planu beležimo, imamo v programu možnost vpogleda v zgodovino posameznih aktivnosti. Pri izboru vrstice v terminskem planu lahko s pritiskom na prikažemo zgodovino izbrane aktivnosti. Za vsako spremembo je prikazan datum, čas, opis ter ali jo je opravil uporabnik ali program sam.



GUID	Kaj se je spremenilo	Iz	Na	Avtomatično
AAA2	connections	Null	1 FS (0)	NE
AAA2	startDate	19.5.2010	25.5.2010	DA
AAA3	startDate	19.5.2010	25.5.2010	DA
AAA4	startDate	22.5.2010	28.5.2010	DA

Slika 7: Pregled zgodovine aktivnosti, s prikazom ročnih odnosno avtomatičnih sprememb

4.3.4 Shranjevanje projektov

Za uporabo programa se je potrebno z uporabniškim imenom in geslom povezati s strežnikom, kjer so shranjeni vsi projekti. Uporabnik lahko shrani poljubno projektov, shranjeni projekti pa se hranijo v MySQL bazi podatkov. Tako so vsi projekti vedno shranjeni na strežniku, kjer je postavljen program za izdelavo terminskih planov, uporabniki pa lahko do svojih projektov dostopajo z vsakega računalniškega sistema povezanega na internet.

Poleg zapisa vseh podatkov, ki so bili v terminskem planu vpisani, se pri shranjevanju projekta shrani tudi celotna zgodovina upravljanja. Shranjevanje zgodovine ohrani tudi funkcionalnost "Undo–Redo".

Vsak projekt z vsemi podatki, vključno z zgodovino sprememb se lahko shrani kot XML datoteka.

4.3.5 Dodatek stroškov terminskem planu

Terminski plan je urnik aktivnosti in povezav med njimi, vsaki aktivnosti pa lahko dodamo strošek kot dodatni parameter. To omogoča, da prek programa izdelamo:

- diagram porazdelitve stroškov med aktivnostmi;
- diagram časovne porazdelitve stroškov glede na izdelani terminski plan.

Poleg tega implementacija stroškov znotraj terminskega plana omogoča, da v nadaljevanju na podlagi terminskega plana izdelamo tudi 5D simulacijo gradnje objekta v programu Sketchup.

4.4 Implementacija

Program za izdelavo terminskih planov, je kompleksna aplikacija in bi opis celotnega delovanja programa presegel namen diplomske naloge. V nalogi bomo predstavili le nekaj ključnih idej in rešitev katere smo uporabili.

Najprej se je bilo potrebno odločiti, kako bomo v programu hranili vse podatke potrebne za delovanje terminskega plana. Flex ponuja hranjenje podatkov v spremenljivki tipa `ArrayCollection`, katero si lahko predstavljamo kot polje z nekaterimi dodatnimi zmožnostmi kot so: lažje dodajanje, brisanje in razvrščanje elementov, ter obveščanja o spremembah spremenljivke ostalim objektom programa.

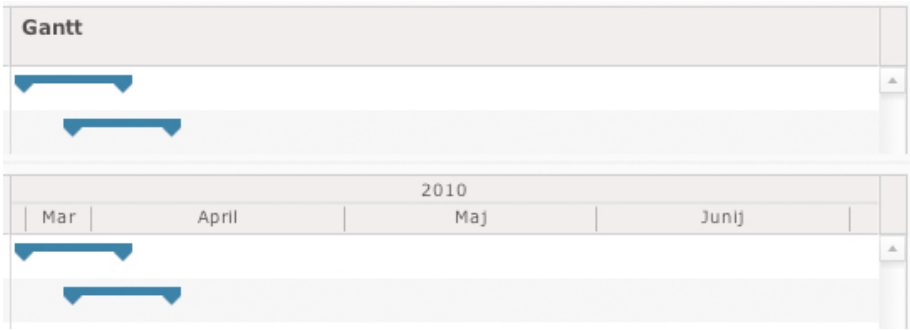
Zapisovanje podatkov v spremenljivki tipa `ArrayCollection` je zelo primerna tudi za prikazovanje podatkov v tabeli `DataGrid`. Celoten prikaz in delovanje terminskega plana temelji na obdelavi podatkov zapisanih v `ArrayCollection`-u, ter prikazu teh podatkov v tabeli `DataGrid`.

`DataGrid` v celicah tabele prikazuje le tekstovne podatke, omogoča pa spremembo izgleda posamezne celice, glede na podatke, ki so zapisani v spremenljivki, ki je vir podatkov za tabelo. Za vsako celico sta na voljo dve metodi spremembe izgleda in sicer `itemRendered` in `itemEditor`, za izgled glave tabele pa skrbi `headerRenderer`.

`itemRenderer` omogoča, da izgled v celici tabele, pogled poljubno spremenimo. Pri implementaciji tabele za prikaz terminskega plana smo uporabili `itemRenderer` za prikaz datuma začetka ter zaključka aktivnosti, Ganttovih diagramov ter aktivnosti s potrebnim zamikom glede na hierarhični nivo.

`itemEditor` se vklopi, če je v `DataGrid`-u za celico definirano, da je mogoča sprememba podatkov. Uporabljen je bil za spreminjanje imen, trajanja in odstotka opravljenega dela aktivnosti.

`headerRenderer` deluje kot `itemRenderer` le s to razliko, da spreminja izgled glave tabele. Uporabili smo ga za prikaz datumov v glavi stolpca, kjer se prikazujejo Ganttovi diagrami. Na sliki 8, sta prikazana izgleda glave tabele brez in z uporabo `headerRenderer`-ja.



Slika 8: Sprememba izgleda glave tabele z uporabo `headerRenderer`-ja

4.4.1 Hierarhični nivoji

Hierarhične nivoje v terminskem planu formiramo s premikom ene aktivnosti pod drugo, tako, da je aktivnost višjega nivoja zajeta znotraj aktivnosti nižjega nivoja. Za lažje razumevanje hierarhičnih nivojev v terminskem planu si oglejmo sledeči primer.

V preglednici 1 imamo zapisanih nekaj aktivnosti ter njihove hierarhične nivoje. Vodoravni premik aktivnosti omogoča lažji vpogled v hierarhijo aktivnosti. Znotraj aktivnosti A so definirane aktivnosti B, C ter D, kar pomeni, da aktivnost A zajema vse njej podrejene aktivnosti. Datumi začetka aktivnosti B, C ter D ne morejo biti bolj zgodnji kot začetni datum aktivnosti A, enako velja za datume zaključka, saj se aktivnosti B, C in D ne morejo končati po zaključku aktivnost A. Program pri izdelavi terminskega plana sam skrbi, da bodo datumi aktivnosti pravilni glede na njihovo hierarhično razporeditev.

Preglednica 1 - Primer hierarhične razporeditve aktivnosti

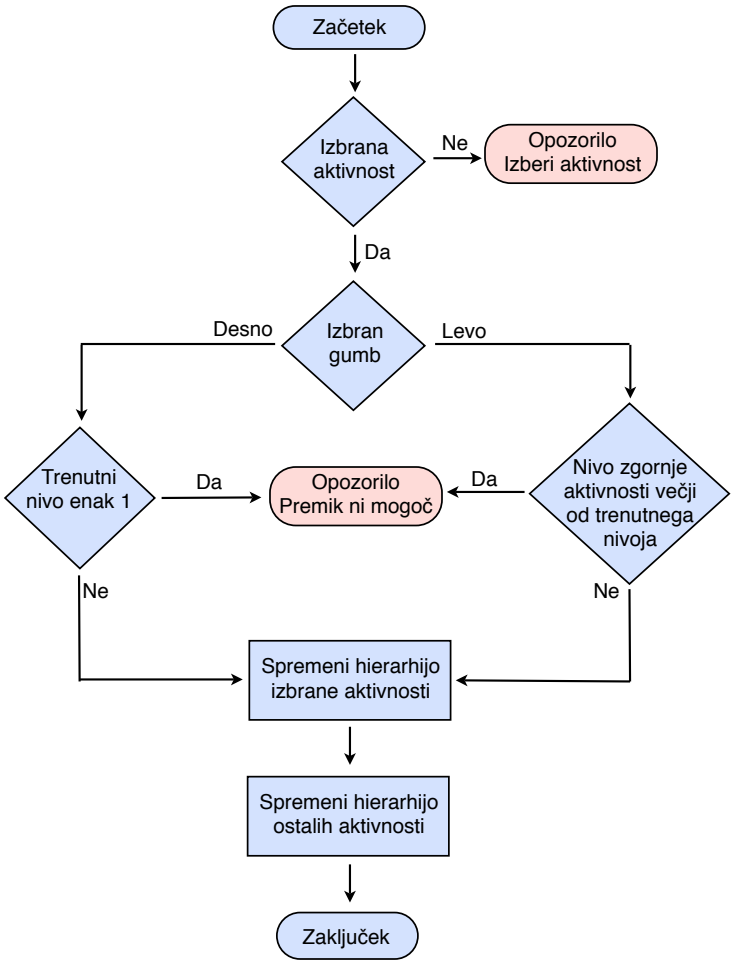
Aktivnosti	Hierarhični nivo
Aktivnost A	1
Aktivnost B	2
Aktivnost C	2
Aktivnost D	3
Aktivnost E	1
Aktivnost F	2

Aktivnosti s hierarhičnem nivojem 1, imenujemo glavne aktivnosti in jih v terminskem planu označimo s krepko pisavo. V terminskem planu aktivnosti hierarhično premikamo z gumboma  ter  .

Na sliki 9 vidimo potek funkcije `WBSSwitch`, ki je poklicana, z izborom gumba za spremembo hierarhije.

Funkcija `WBSSwitch` najprej preveri ali je aktivnost v terminskem planu izbrana, če ni, se izvajanje ustavi, prikaže pa se ustrezno opozorilo. Funkcija deluje ločeno za premik aktivnosti za hierarhični nivo navzgor in navzdol.

Nato preveri ali je za dano aktivnost izbrani premik mogoč. Če ni mogoč se izvajanje funkcije ustavi z opozorilom. V kolikor je premik mogoč, funkcija `WBSSwitch` spremeni hierarhični nivo izbrane aktivnosti in ustrezno spremeni tudi hierarhijo ostalih aktivnosti.



Slika 9: Potek funkcije `WBSSwitch` za spreminjanje hierarhije

Implementacija funckije `WBSSwitch` je navedena v prilogi C.

4.4.2 Pregled povezav aktivnosti

Ena izmed zahtevnejših nalog pri pisanju programa je bila izvedba povezav med aktivnostmi. Na voljo imamo tri možnosti povezovanja aktivnosti in sicer FS, FF ter SS; za vsako povezavo lahko določimo tudi zamik, za vsako aktivnost pa lahko definiramo več povezav z drugimi aktivnostmi.

Aktivnosti se v terminski plan vpisujejo, kot prikazuje slika 10. Najprej je zapisana zaporedna številka aktivnosti—ID, s katero želimo dano aktivnost povezati, sledi tip povezave ter v oklepaju zapisan morebitni zamik povezave (časovni zamik znotraj oklepaja podajamo v dnevih).

Slika 10: Vpis povezave v terminskem planu

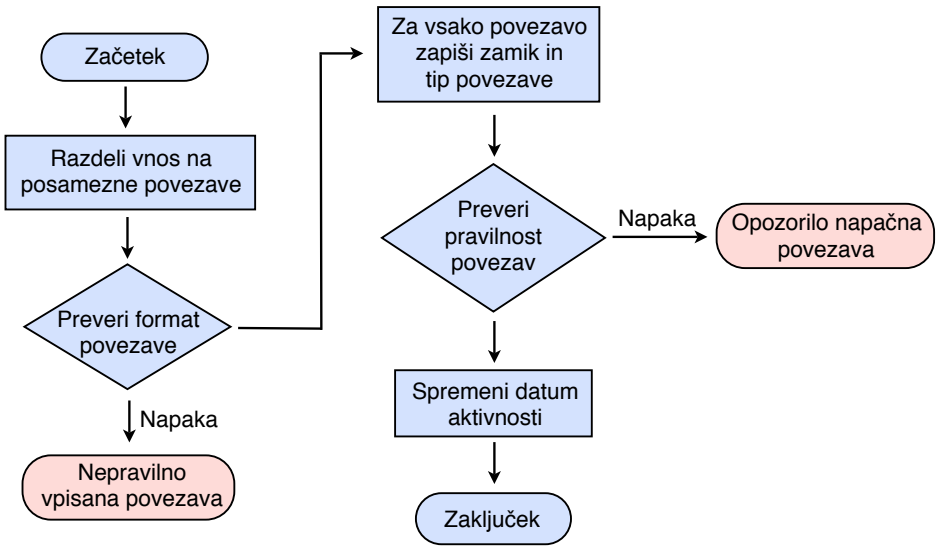
Primer povezave znotraj terminskega plana je: 7 FS (3), ki dano aktivnost poveze z aktivnostjo z ID-jem 7, tip povezave je FS, zamik pa 3 dni. Če izbrano aktivnost povezujemo z večimi aktivnostmi, te ločimo z vejicami. Če tip povezave ni vpisan program privzame povezavo tipa FS. Oglejmo si še primer povezovanja z večimi aktivnostmi:

3 FF (4), 2, 8 SS, 5 (3) – program interpretira povezavo kot 3 FF (4), 2 FS (0), 8 SS (0), 5 FS (3)

Po vpisu povezav za aktivnost program najprej razdeli vnos na posamezne povezave, ki so med seboj ločene z vejicami, nato preveri pravilnost zapisa povezav. Če nek format ni pravilen, program uporabnika na napako opozori.

Nato program za vsako povezavo zapiše tip ter zamik in preveri veljavnost, implementacija preverjanja povezav je opisana v naslednjem poglavju. V primeru neveljavne povezave program o napaki opozori uporabnika, v nasprotnem primeru program izbere kritično povezavo, to je tisto, ki

za aktivnost pomeni najkasnejši datum zaključka in aktivnosti ustrezno spremeni datum začetka in zaključka. Postopek je prikazan na sliki 11.



Slika 11: Kontrola vpisanih povezav

Implementacija kontrole vpisanih povezav je prikazana v prilogi D.

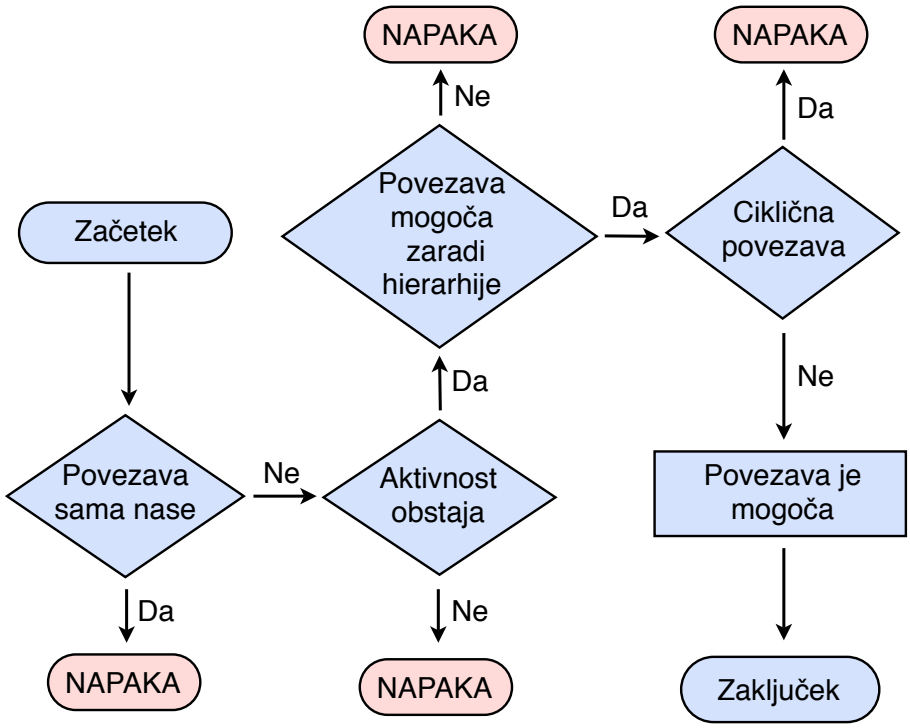
4.4.3 Pregled veljavnosti povezave

Za vsako vpisano povezavo je potrebno preveriti ali je povezava veljavna, primeri neveljavnih povezav so:

1. povezovanje aktivnosti same nase;
2. povezovanje z neobstoječo aktivnostjo;
3. neveljavno povezovanje zaradi hierarhije aktivnosti
4. ciklične povezave

Funkcija `checkConnection` v programu preveri veljavnosti povezav v terminskem planu, njen potek je prikazan na sliki 12, implementacija pa je navedena v prilogi E.

Program najprej preveri ali povezava ne kaže sama na sebe, tovrstna povezava bi bila nesmiselna. Nato preveri ali vpisana aktivnost obstaja v termiskem planu, sicer se preverjanje zaključi. Sledita kontroli za preverjanje povezave glede na hierarhijo ter preverjanje cikličnih povezav. V kolikor sta kontroli uspešni se preverjanje dane povezave uspešno zaključi, povezava pa je mogoča.



Slika 12: Pregled veljavnosti povezave

5 PROGRAM ZA SIMULACIJE GRADNJE

Drugi del naloge je osredotočen na razvoj programa za izdelavo integriranih modelov in simulacije gradnje objektov. Kot je že omenjeno v uvodu, smo uporabili Sketchup, kot program za modeliranje.

Celotna simulacija gradnje objektov je opravljena znotraj programa Sketchup z uporabo programskega vmesnika Sketchup Ruby API.

5.1 Zahteve

Pri programu za izdelavo integriranih modelov in simulacij gradnje smo upoštevali naslednje zahteve:

Funkcionalne zahteve

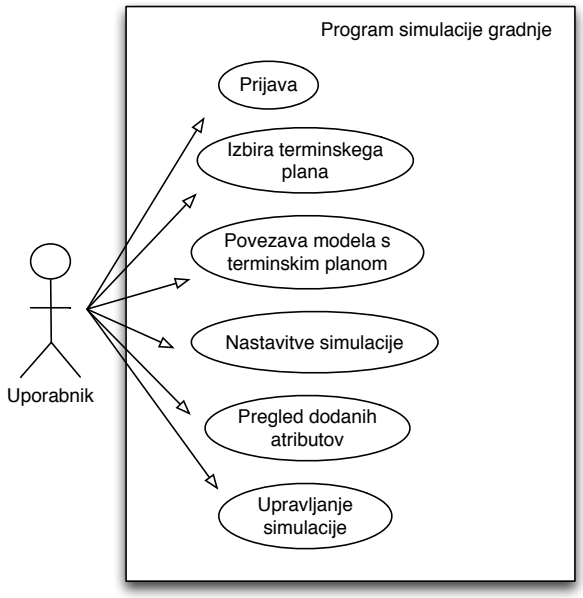
- možnost povezovanja elementov 3D modela z aktivnostmi v terminskem planu;
- nastavitve časa simulacije gradnje;
- upravljanje predvajanja simulacije;
- prikaz grafa stroškov glede na čas;
- prikaz aktivnosti, ki se tekom simulacije izvajajo;
- prikaz barv aktivnosti, ki se nahajajo v 3D modelu;
- možnost, da se 3D element po koncu izvajanja aktivnosti skrije;
- pregled zapisanih atributov 3D elementov (v kolikor so bili definirani).

Tehnične zahteve

- uporaba brezplačnega programa za 3D modeliranje
- integracija povezovanja 3D elementov in terminskega plana znotraj Sketchup-a
- enostavna izdelava simulacije

5.2 Diagram uporabe

Podobno kot pri programu za izdelavo terminskih planov, smo pred začetkom implementacije programa za izdelavo simulacije gradnje izdelali diagram uporabe, kjer so na sliki 13 prikazane možnosti, ki jih bo imel na voljo uporabnik pri uporabi programa.



Slika 13: Diagram uporabe programa za simulacije gradnje

5.3 Zasnova

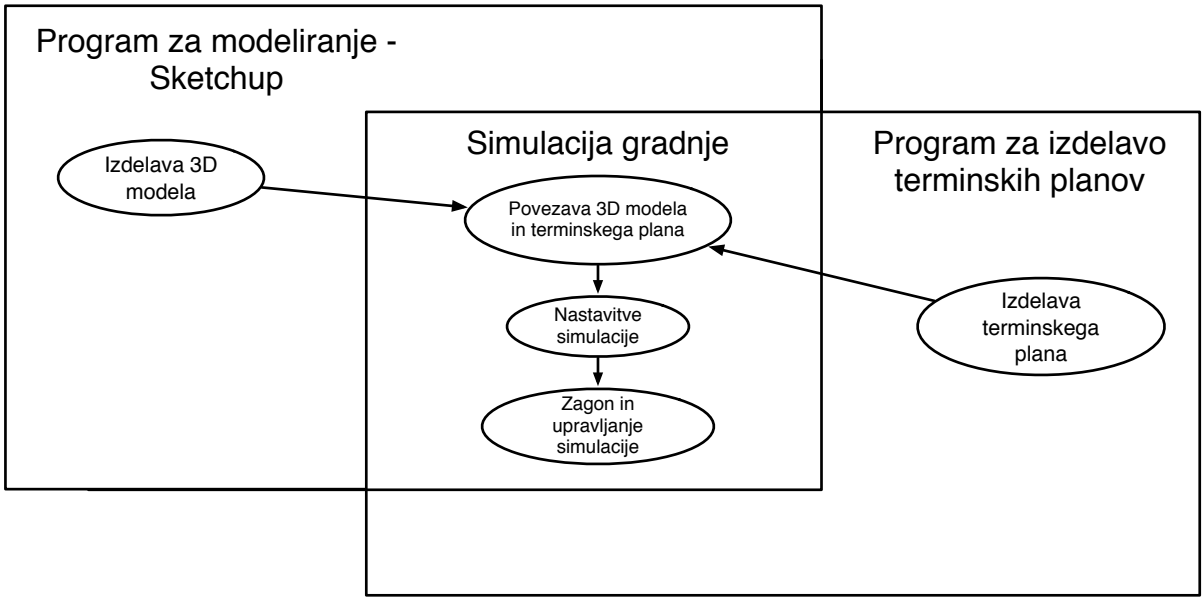
Program za izdelavo terminskega plana je izdelan z ogrodjem Flex, ki se izvaja v brskalniku. Sketchup omogoča, da se znotraj programa, kot mogoči uporabniški vmesnik uporabi kar okno brskalnika. To nam omogoča, da za upravljanje s programom ter za prikaz vseh podatkov, uporabimo kar ogrodje Flex. Tako program poteka na dveh platformah. Za komunikacijo med Sketchup-om ter Flex-om skrbi skriptni jezik JavaScript, kot je predstavljeno na sliki 14.

Komunikacija je dvosmerna, tako lahko program znotraj brskalnika komunicira s Sketchup-om, in

Slika 14: Komunikacija med Sketchup-om ter aplikacijo v Flex-u

obratno. Ker med Flex platformo in Sketchup-om ni mogoče postaviti direktne povezave, se kot "vmesnik" uporabi JavaScript.

Izdelava integriranega modela znotraj programa Sketchup je omogočena s povezavo 3D modela in terminskega plana. Gre za interoperabilnost dveh programov, saj znotraj programa Sketchup uporabimo podatke pridobljene iz programa za terminske plane, kot je prikazano na sliki 15.



Slika 15: Interoperabilnost dveh programov za izdelavo integriranega modela

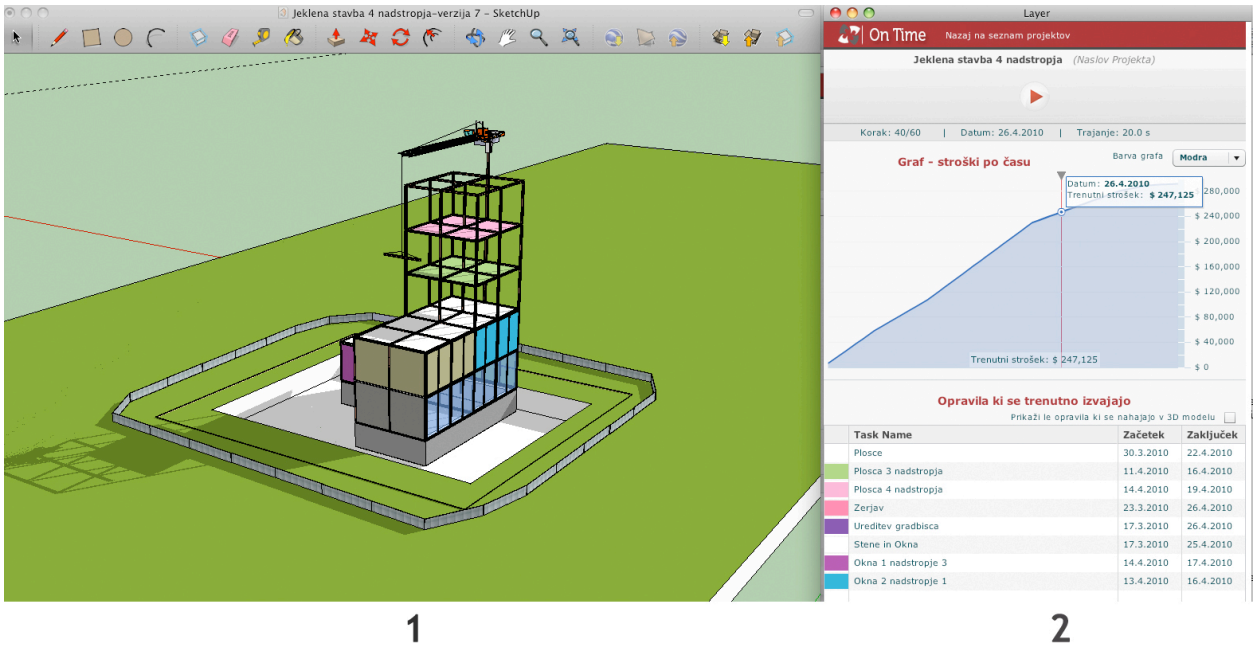
Program se izvaja v treh korakih:

1. ***Povezava modela s terminskim planom:*** prvi korak povezuje elemente 3D modela z aktivnostmi v terminskem planu, s tem 3D elementom dodamo dodatne atribute kot sta čas izvedbe ter strošek.

2. ***Nastavitev simulacije:*** drugi korak, je preprost, saj zahteva le določitev dolžine trajanja simulacije od 5–120 sekund.
3. ***Pregled in upravljanje simulacije:*** v tretjem koraku imamo možnost upravljanja simulacije gradnje objekta. Prikazani so tekoči stroški ter seznam vseh izvajanih aktivnosti.

5.3.1 Uporabniški vmesnik

Uporabniški vmesnik smo postavili ob desni rob okna 3D modela. Na sliki 16 vidimo izgled aplikacije med simulacijo: model zgradbe med gradnjo je označen z (1), uporabniški vmesnik pa z (2); kjer se poleg funkcije za nadaljevanje ali ustavljanje simulacije prikazuje tudi graf stroškov (v kolikor so bili stroški vpisani v terminski plan), spodaj pa je še spisek vseh aktivnosti, ki se v terminskem planu trenutno izvajajo. Aktivnosti v 3D modelu imajo pred imenom prikazano tudi barvo, s katero so v modelu označeni, poleg tega sta prikazana tudi datuma začetka in zaključka aktivnosti.



Slika 16: Program za simulacijo gradnje znotraj Sketchup-a

5.3.2 Pregled in upravljanje simulacije

Zadnji korak pri uporabi programa je upravljanje simulacije gradnje. Na sliki 17 je prikazan uporabniški vmesnik pri zagonu simulacije in prikazuje:

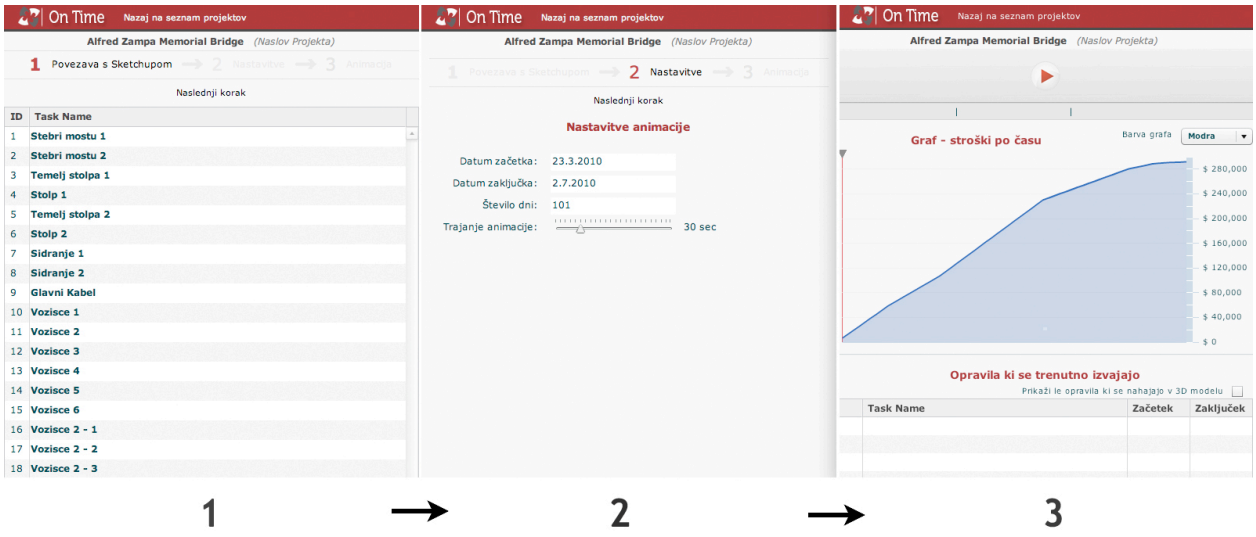
1. Kontrolo procesa simulacije, s tipko za zagon in zaustavitev. Prikaz tekočega datuma ter trajanja simulacije v sekundah.
2. Graf stroškov v odvisnosti od časa; za vsak korak simulacije, sta prikazana tudi trenutni strošek ter datum.
3. Tabelo aktivnosti, ki se trenutno izvajajo. Za aktivnosti, ki so prisotne v simulaciji, se prikaže tudi barva, ki je vidna v 3D modelu. Možen je tudi prikaz le tistih aktivnosti, ki se nahajajo v 3D modelu.

Slika 17: Uporabniški vmesnik

5.4 Implementacija

Uporabniški vmesnik ima svoje okno, kjer Sketchup naloži privzeti brskalnik operacijskega sistema. Tako je bila povezava uporabniškega vmesnika znotraj Sketchup-a in programa za izdelavo terminskih planov preprosta, saj obe aplikaciji poganja brskalnik.

Program za simulacijo gradnje se v Sketchup-u aktivira z izborom vtičnika (angl., *plugin*) v meniju Plugins. Po zagonu se je potrebno vpisati z uporabniškim imenom ter geslom, nakar se prikažejo vsi projekti povezani z vpisanim uporabniškim imenom. Po izboru projekta se delo začne s povezavo 3D modela z izbranim terminskim planom, kar je opisano v naslednjem poglavju.



Slika 18: Trije koraki uporabniškega vmesnika

5.4.1 Povezava 3D modela s terminskim planom

Začetni korak pri uporabi programa za izdelavo simulacije gradnje je povezava 3D modela s terminskim planom. V kolikor želimo izdelati simulacijo gradnje moramo programu sporočiti kateri 3D element pripada kateri aktivnosti v terminskem planu. Odločili smo se, da bomo elemente po uspešnem postopku povezovanja premaknili v posebni sloj (angl., *layer*) znotraj programa

Sketchup. To omogoča, da bomo v programu brez težav prikazali ali skrili vse elemente, ki so povezani z isto aktivnostjo v terminskem planu. Ime za sloj se bo tvoril po sledečem načelu: v oklepaju bo zapisan GUID (angl., *Global Unique Identifier*), sledilo pa bo ime aktivnosti iz terminskega plana. Primer imena sloja je: (AAA1) Temeljenje.

Vsakemu 3D elementu, ki bo uspešno povezan s terminskim planom, bodo dodani parametri zapisani v preglednici 2.

Preglednica 2 - Pomen in oznake parametrov pripisanih 3D elementom

Parameter	Pomen
taskName	Ime aktivnosti v terminskem planu
GUID	Globalna unikatna številka aktivnosti
UID	Zaporedna številka aktivnosti
startDate	Datum začetka aktivnosti
finishDate	Datum zaključka aktivnosti
duration	Trajanje aktivnosti v dnevih
percentComplete	Odstotek opravljenega dela
hideAtEnd	Ali se element na koncu aktivnosti skrije

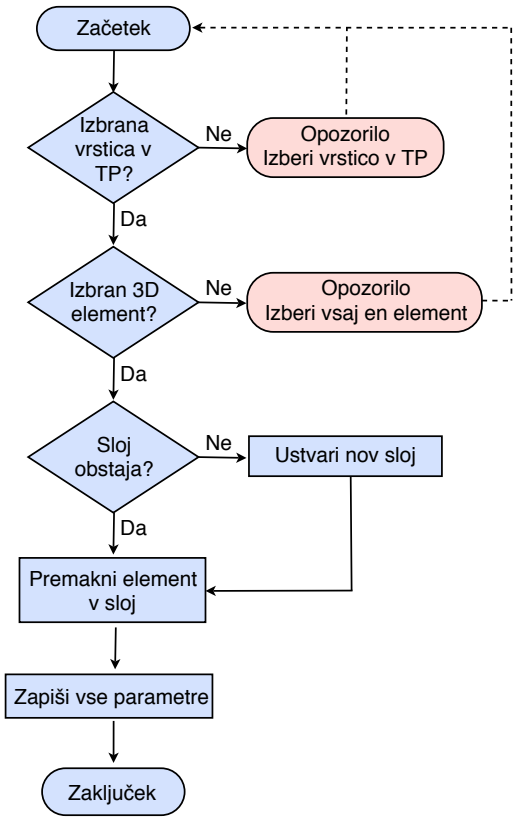
Poleg parametrov, ki so direktno vezani s terminskim planom, si za vsak 3D element zapišemo tudi podatke o materialu. Podatke o materialu moramo poznati saj se tekom simulacije barva elementa spremeni, ko je aktivnost, s katerim je 3D element povezan, v teku. Vsaka aktivnost v terminskem planu je v 3D modelu prikazana z naključno izbrano barvo. S tem, omogočimo, da se med simulacijo barvno ločijo elementi, ki so že zgrajeni, ter tistimi, ki se še gradijo.

Sketchup ima za vsak element zapisanih kar nekaj podatkov o materialu, kot zaporedna številka, barva, prosojnost, tip, tekstura in drugo. Za vsak element, kjer je definiran material, si zapišemo parametre iz preglednice 3. Poleg parametrov naštetih v preglednici, za 3D element tipa “face” (ploskev), si dodatno zapišemo še parametre backColor, backAlpha, backMaterial_id, backMaterial_type, saj ima ploskev ločeno definiran material na sprednji in hrbtni strani.

Preglednica 3 - Pomen in oznake parametrov za material 3D elementa

Parameter	Pomen
color	Barva
alpha	Prosojnost
material_id	Zaporedna številka
material_type	Tip

Sedaj opišimo postopek povezovanja 3D elementa z aktivnostjo v terminskem planu prikazan na sliki 19. Program najprej preveri ali je bila izbrana vrstica v terminskem planu, če ni uporabnika opozori, da mora izbrati aktivnost v terminskem planu. V naslednjem koraku, program preveri, če je uporabnik izbral vsaj en 3D element, katerega lahko poveže s terminskim planom, čene uporabnika na napako opozori.



Slika 19: Potek povezave 3D elementov in terminskega plana

Ko imamo izbrane 3D elemente in aktivnost v terminskem planu, pregledamo ali za aktivnost že obstaja pripadajoči sloj, katerega ime se tvori kot je navedeno v prvem odstavku. V kolikor sloja še ni, ga program ustvari ter 3D elemente vanj premakne. Nato za vsak element zapišemo parametre, ki so omenjeni v preglednici 2 in 3, s tem se povezava elementa ter terminskega plana zaključi.

Med povezovanjem elementov 3D modela in terminskega plana program zapiše tudi vse aktivnosti, ki so bile z modelom uspešno povezane, kar omogoča, da pri simulaciji gradnje (3 korak) pregledamo le podatke relevantne za uspešen prikaz simulacije. Zgodi se namreč lahko, da nekatere aktivnosti v terminskem planu, niso povezane s 3D modelom; te aktivnosti lahko pri simulaciji ignoriramo (prikažemo jih le v seznamu trenutno izvajanih aktivnosti) in se izognemo računu, ki je potreben za aktivnosti, ki so z modelom povezane.

Omeniti je potrebno, da vsi s terminskim planom uspešno povezani elementi postanejo nevidni, saj so njihovi sloji nastavljeni tako. To olajša delo, saj ostanejo prikazani samo elementi, kateri še niso povezani. V Sketchup-u, je mogoče sloje prikazati ali skriti prek dialoga, ki ga vklopimo z Windows–Layers.

Dodatna funkcija, ki je nastavljena pri zagonu programa pa je Sketchup-ov prikaz vseh 3D elementov po barvah slojev, katerim pripadajo. V uporabniškem vmesniku lahko s klikom na “Pobarvaj po sloju/aktivnosti” prikažemo elemente v dodeljenih barvah ali v barvah njihovih slojev.

5.4.2 Nastavitve simulacije

Ko vse 3D elemente v Sketchup-u uspešno povežemo z aktivnostmi v terminskem planu, lahko nadaljujemo delo z nastavitvami simulacije. Tu se prikaže datum začetka ter zaključka terminskega plana, izberemo pa lahko dolžino simulacije od 5–120 sekund.

5.4.3 Priprava podatkov za simulacijo

Prvi problem, pri simulaciji gradnje v programu Sketchup je bila implementacija simulacij znotraj programa. Simulacijo gradnje smo razdelili na več enakih časovnih intervalov. Časovni interval bomo imenovali korak in ga omejili tako, da ne sme biti krajši od 0.1 sekunde. Veliko število korakov v kratkem času simulacije bi lahko privedlo do težav z odzivnostjo programa.

Med koraki je mogoče opraviti spremembe v 3D modelu. Ko so vsi podatki pripravljeni, za celotni terminski plan vemo, katera izmed naslednjih sprememb se bo odvijala v posameznem koraku:

1. **Začetek aktivnosti**: vsi elementi, povezani z aktivnostjo, se prikažejo ustrezno pobarvani.
2. **Konec aktivnosti**: vsi elementi, povezani z aktivnostjo, prevzamejo barve, s katerimi so bili modelirani.

V drugem koraku uporabnik izbere željeno trajanje simulacije, program najprej izračuna razmerje med izbranim trajanjem simulacije (`animationDuration`) ter številom dni v terminskem planu (`durationInDays`), izračunano razmerje program zapiše v spremenljivko `timerStep`, kar je trajanje enega koraka (v sekundah).

$$\texttt{timerStep} = \frac{\texttt{animationDuration}}{\texttt{durationInDays}}$$

Vrednost spremenljivke `timerStep` v naslednjem koraku po potrebi popravimo tako, da je najmanj 0.1. Potrebno število korakov (`nrOfFrames`) izračunamo kot kvocient izbranega trajanja simulacije ter popravljene vrednosti spremenljivke `timerStep`. Izračunamo še koliko dni v terminskem planu traja en korak simulacije, rezultat se zapiše v spremenljivko `nrDaysPerFrame`.

$$\texttt{nrOfFrames} = \frac{\texttt{animationDuration}}{\texttt{timerStep}}$$

$$\texttt{nrDaysPerFrame} = \frac{\texttt{durationInDays}}{\texttt{nrOfFrames}}$$

Primer: Izbrano trajanje simulacije je 20 sekund, število dni v terminskem planu je 283. Izračunamo $\texttt{timerStep} = 20/283 = 0.071$, vrednost `timerStep` popravimo na 0.1, ter dobimo $\texttt{nrOfFrames} = 20/0.1 = 200$ in $\texttt{nrDaysPerFrame} = 283/200 = 1.42$.

V nadaljevanju program zahteva vse aktivnosti, katere smo zapisali pri prvem koraku, ko smo povezovali terminski plan in 3D model. V kolikor tega podatka ne dobimo, program javi napako, simulacija pa se ne izvede. Povezane aktivnosti se hranijo v polju `layersArr`.

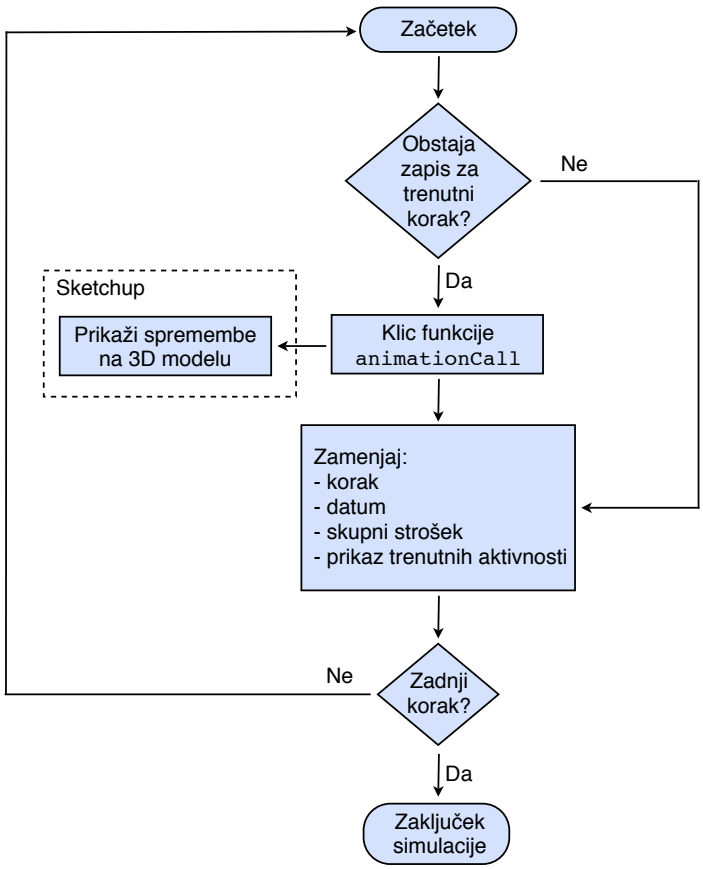
Sledi klic funkcije `populateDataForAnimation`, ki na osnovi polja `layersArr` pripravi podatke za:

1. vsak korak simulacije, kjer se v polje `animationArr` zapišejo podatki o začektu in koncu vsake aktivnosti povezane s 3D modelom,
2. potek stroškov tekom simulacije in rezultate shrani v spremenljivko `moneyAC`, kjer se za vsak korak zapiše datum simulacije, strošek v posameznem koraku, ter skupni strošek,
3. spisek vseh aktivnosti, s podatki o korakih, kjer se aktivnost začne in konča. Zapišejo se v spremenljivko `inProgressAC`, ki tekom simulacije prikazuje izvajane aktivnosti.

Z uspešnim zaključkom funkcije `populateDataForAnimation` se konča priprava podatkov za simulacijo, implementacija funkcije je prikazana v prilogi A.

5.4.4 Zagon in upravljanje simulacije

Program registrira merilnik časa (angl., *timer*), simulacija pa se lahko prične. Merilnik časa v časovnih intervalih dolžine `timerStep` v programu pokliče funkcijo `frameEnter`, katere delovanje je prikazano na sliki 20.

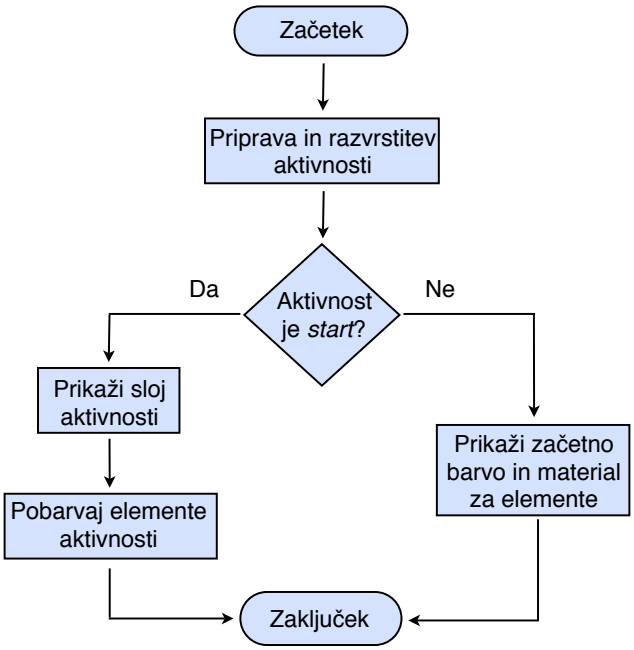


Slika 20: Funkcija `frameEnter`

Funkcija `frameEnter` bo priklicana za vsak korak simulacije, izračunano število korakov smo zajeli v spremenljivki `nrOfFrames`. Funkcija najprej preveri ali v polju `animationArr` za trenutni korak obstaja zapis aktivnosti za prikaz v 3D modelu. Zapis za vsako aktivnost je sestavljen iz treh delov in sicer:

1. type: obeleži da gre za začetek (start) ali zaključek (finish) aktivnosti;
2. layerName: ime aktivnosti ali sloja, sestavljenega iz unikatne zaporedne številke ter imena aktivnosti;
3. color: barva aktivnosti za prikaz v 3D modelu, zapisana v formatu RGB.

V vsakem koraku je lahko zapisanih več aktivnosti. Program podatke o aktivnostih posreduje Sketchup-u s funkcijo `animationCall`, ki nato stori vse potrebno za pravilen prikaz 3D elementov v modelu; funkcija je predstavljena na sliki 21, njena implementacija pa je v prilogi B.



Slika 21: Funkcija `animationCall`

Sledi prikaz datuma aktivnosti, ki so v teku, ter skupnih stroškov od začetka simulacije do trenutnega koraka. Merilnik časa klic funkcije `animationCall` ponavlja do zadnjega koraka, kjer se simulacija zaključi s prikazom vseh 3D elementov in slojev v modelu.

6 UPORABNIŠKI PRIROČNIK

6.1 Program za izdelavo terminskih planov

Uporabo programa bomo predstavili tako, da uporabnika popeljemo skozi celoten postopek izdelave terminskega plana. Ogledali si bomo izdelavo terminskega plana za tri nadstropni objekt.

Za uporabo programa mora biti na brskalniku nameščen Adobe Flash Player, katerega dobimo brezplačno na naslovu *http://get.adobe.com/flashplayer/*.

Brskalnik usmerimo na spletni naslov *http://5d.deskriptor.si/ontime/tp/slo/*.

V program se prijavimo z uporabniškim imenom in geslom, polja so že zapolnjena s podatki za upravljanje s preiskusnim računom, zato lahko nadaljujemo s pritiskom na gumb "Naprej!".

Po uspešni prijavi v sistem se na zaslonu prikažejo vsi projekti, ki so za vpisano uporabniško ime na voljo. Ker bomo ustvarili nov terminski plan pritisnemo gumb "Odpri nov projekt", ko nas program vpraša po imenu projekta, vpišemo *"3-nadstropni objekt"* in pritisnemo "OK".

Oglejmo si najprej orodja, ki so na voljo za upravljanje s terminskim planom (z leve proti desni);

 – Dodaj novo aktivnost v terminski plan

 – Premik aktivnosti po hierarhiji za nivo navzdol

 – Premik aktivnosti po hierarhiji za nivo navzgor

 – Undo–Razdri zadnje

 – Redo–Razdrto povrni

 – Prikaz zgodovine za izbrano aktivnost

 – Prikaži/skrij stroške v terminskem planu

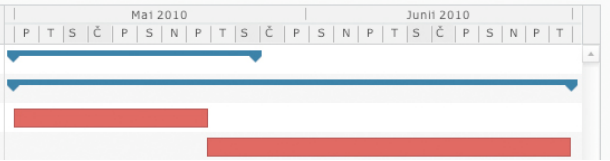
V terminski plan je avtomatično vpisana ena aktivnost. Spremenimo ime aktivnosti v *Priprava gradbišča*, spremenimo trajanje aktivnosti na 10 dni. Z izborom gumba dodamo terminskemu planu novo aktivnost, katero imenujemo *Temeljenje*, nato dodamo še dve aktivnosti, *Izkop gradbene jame* ter *Izvedba temeljev*. Obe aktivnosti želimo hierarhično podrediti aktivnosti *Temeljenje*. To storimo tako, da aktivnost izberemo ter jo s pritiskom na hierarhično premaknemo nivo višje.

Trajanje aktivnosti *Izkop gradbene jame* nastavimo na 8 dni, *Izvedba temeljev* pa na 15 dni. Ker bomo z izvedbo temeljev začeli šele po končanem izkopu gradbene jame bomo aktivnosti povezali s FS povezavo (zaključek–začetek). To storimo tako, da v polje povezave aktivnosti *Izvedba temeljev* vpišemo zaporedno številko aktivnosti *Izkop gradbene jame*, ki je v našem planu 3. Ker je FS privzeti tip povezave, nam ga ni potrebno zapisati, v kolikor bi želeli katero drugo povezavo pa bi zapisali, naprimer; 3 SS, ali pa dodali še zamik z 3 SS (4).

Dodajmo sedaj še stroške za vpisane aktivnosti. S pritiskom na $, se na mestu Ganttovega diagrama prikaže stolpec za vpis stroškov. Za pripravo gradbišča vpišemo strošek 33.000, za temeljenje pa 480.000.

Trenutni izgled terminskega plana si lahko ogledamo na sliki 22.

ID	Aktivnost	Trajanje	Začetek	Zaključek	Povezave
1	**Priprava gradbišča**	10	20.05.2010	30.05.2010	
2	**Temeljenje**	23	20.05.2010	12.06.2010	
3	Izkop gradbene jame	8	20.05.2010	28.05.2010	
4	Izvedba temeljev	15	28.05.2010	12.06.2010	3 FS (0)



Slika 22: Trenutni terminski plan

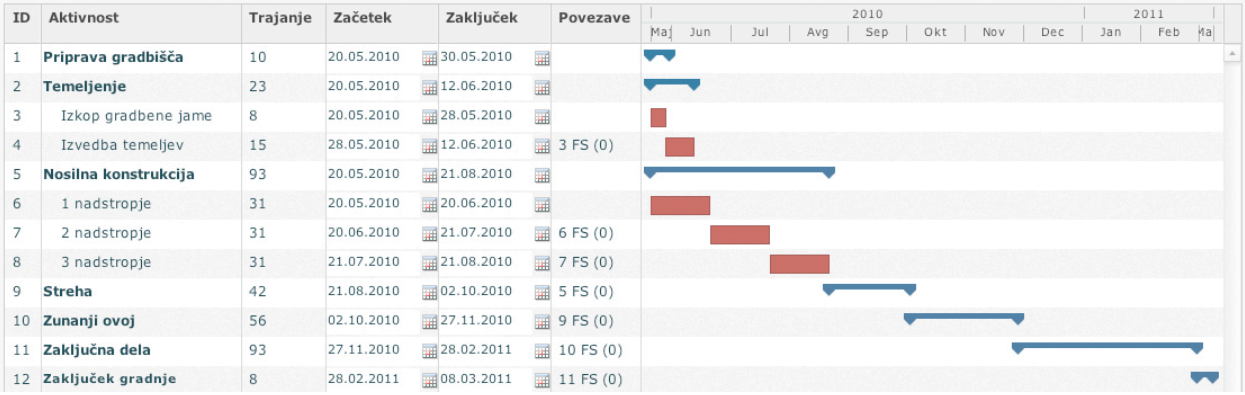
Terminskemu planu dodajmo še aktivnosti, zapisane v preglednici 4.

Preglednica 4 - Aktivnosti za vnos v terminski plan

Ime aktivnosti	Hierarhični nivo	Trajanje	Povezave	Strošek
Nosilna konstrukcija	1	—	—	320.000
1 nadstropje	2	31	—	—
2 nadstropje	2	31	6 FS (0)	—
3 nadstropje	2	31	7 FS (0)	—
Streha	1	42	5 FS (0)	140.000
Zunanji ovoj	1	56	9 FS (0)	900.000
Zaključna dela	1	93	11 FS (0)	485.000
Zaključek gradnje	1	8	11 FS (0)	12.000

Končni izgled terminskega plana je prikazan na sliki 23.

Kot vidimo, program poskrbi, da se glede na vpisane povezave datumi začetka in zaključka vseh aktivnosti uskladijo avtomatično. Terminski plan do sedaj še ni shranjen na strežniku, zato ga shranimo s pritiskom na  . Program se poveže s strežnikom in uporabniku sporoči ali je bilo shranjevanje uspešno.

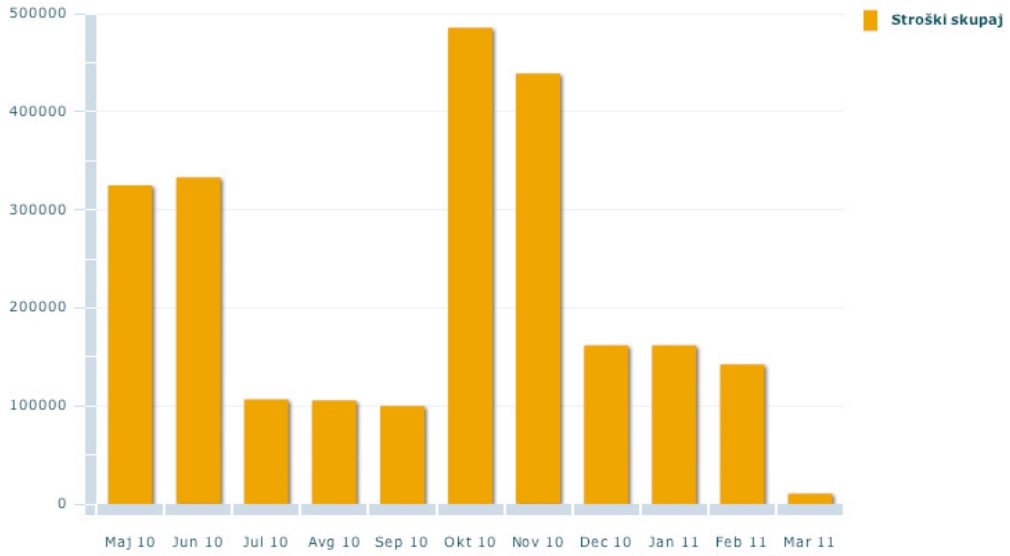


Slika 23: Končni izgled terminskega plana za 3 nadstropni objket

Terminski plan si lahko ogledamo in shranimo tudi v formatu XML in sicer z izborom gumba  . Datoteka XML se prikaže v novem oknu znotraj brskalnika.

Z izborom gumba  si oglejmo analizo stroškov, ki jih ponuja program. Odpre se novo okno, s porazdelitvijo stroškov terminskega plana. Glede na dolžino terminskega plana program izbere

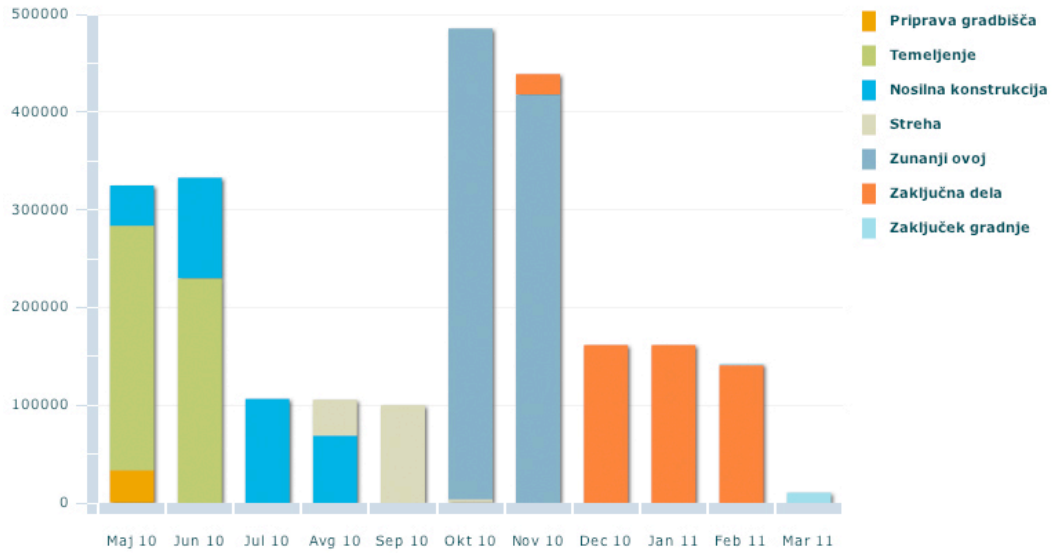
primerni interval za prikaz stroškov; interval je lahko mesec, dva meseca, četrtletje, leto ali dve leti. Naš terminski plan traja 11 mesecev, zato program prikazuje graf porazdelitve stroškov v mesečnih intervalih. Graf stroškov vidimo na sliki 24.



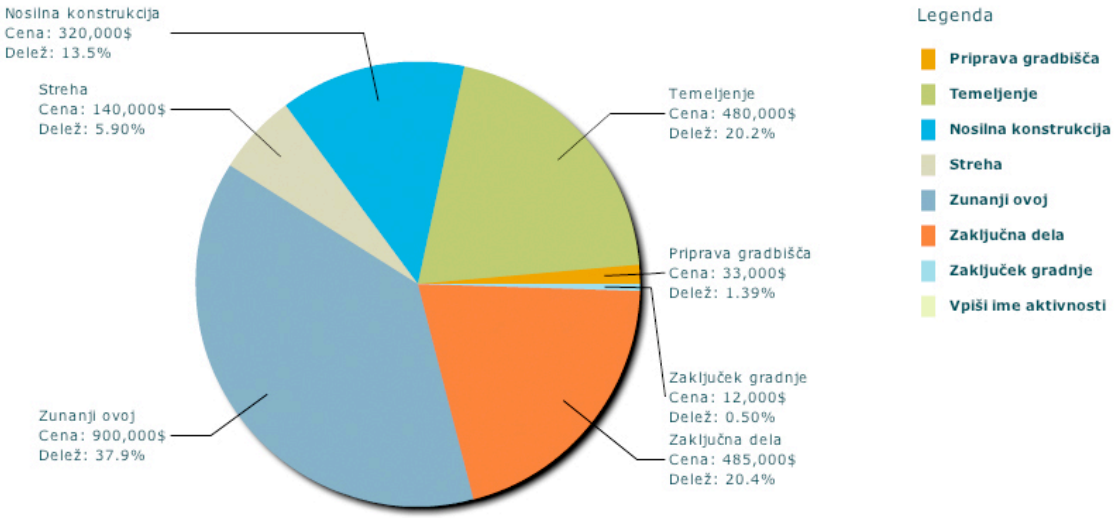
Slika 24: Prikaz porazdelitve skupnih stroškov

Za prikaz porazdelitve stroškov imamo še dodatno možnost, da prikažemo graf, kjer so stroški dodatno porazdeljeni po aktivnostih. V spustnem meniju izberemo "Prikaži posamično", prikaže se enak graf kot prej, le da bodo prikazani prispevki posameznih aktivnosti. Graf vidimo na sliki 25.

Program omogoča še prikaz procentualne porazdelitve stroškov aktivnosti v obliki "torte". Na sliki 26, vidimo porazdelitev stroškov za izdelani terminski plan.



Slika 25: Prikaz porazdelitve skupnih stroškov in prispevkov po aktivnostih



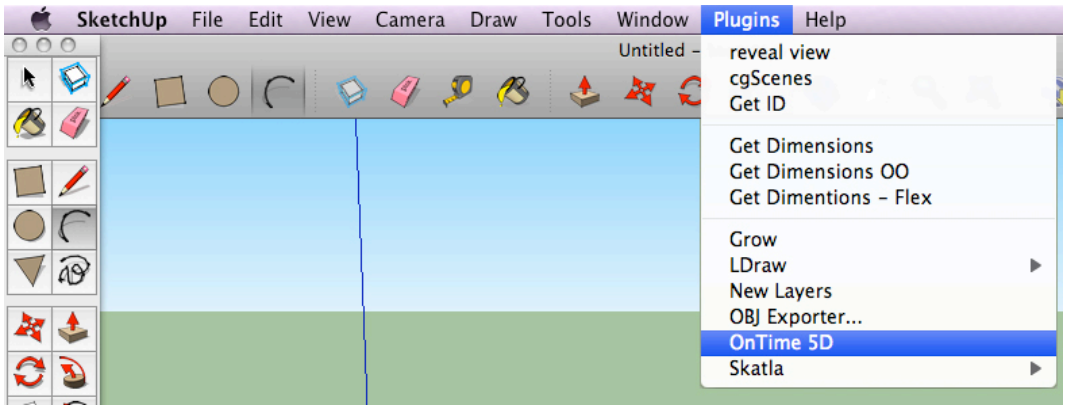
Slika 26: Prikaz porazdelitve štroškov po aktivnostih

6.2 Program za izdelavo simulacije gradnje

Namestitev razširitve v programu Sketchup je zelo preprosto; na kratko bomo predstavili namestitev programa za simulacijo gradnje za operacijska sistema Microsoft Windows ter Mac OS X. Datoteko, ki vsebuje program z imenom `ontime5d.rb` je potrebno kopirati v mapo:

- C:/Program Files/Google/Google SketchUp 7/Plugins za Microsoft Windows;
- Library/Application Support/Google SketchUp 7/SketchUp/Plugins za Mac OS X.

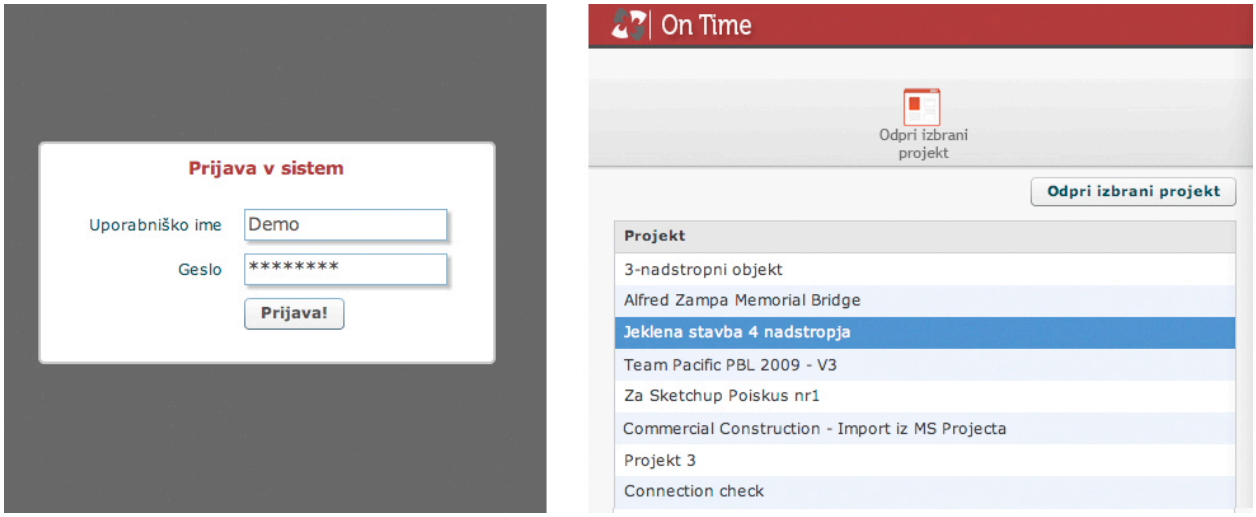
S kopiranjem datoteke v mapo Plugins je namestitev končana; program za simulacijo gradnje bo na voljo pri naslednjem zagonu programa Sketchup in sicer v podmeniju Plugins—OnTime5D, kot je prikazano na sliki 27.



Slika 27: Izbira vtičnika za simulacijo gradnje

Po zagonu programa se znotraj Sketchup-a odpre novo delovno okno, v katero se naloži program za simulacijo gradnje. Priporočljivo je, da se okno Sketchup-a zmanjša toliko da sta obe okni vidni. Prvi korak pri uporabi programa je vpis uporabniškega imena ter gesla, polja sta v privzetem stanju zapolnjena s podatki za preiskusni račun, zato lahko nadaljujemo s klikom na gumb "Prijava!" (slika 28–levo).

Za vpisano uporabniško ime se v programu prikažejo terminski plani, s katerimi bomo v nadaljevanju povezovali 3D elemente, ter kreirali simulacijo gradnje. V seznamu izberemo terminski plan, s pritiskom na gumb "Odpri izbrani projekt" pa z delom nadaljujemo (slika 28–desno).



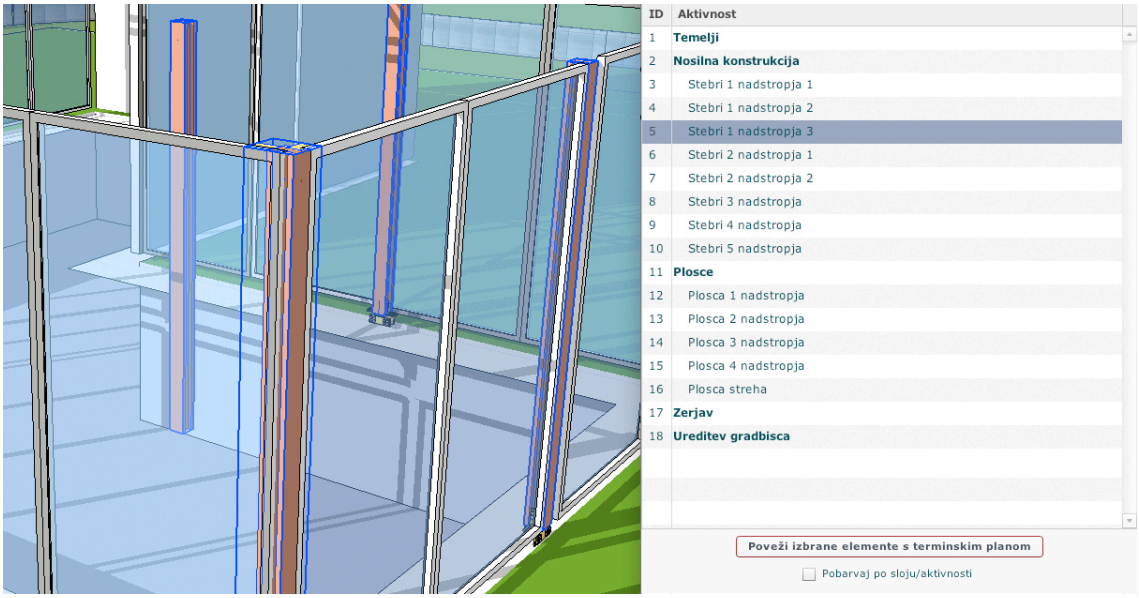
Slika 28: Okni za prijavo (levo) ter izbor terminskega plana (desno)

Po izboru terminskega plana, program prikaže vse aktivnosti, ki so bile za izbrani terminski plan vpisane. Povezovanje 3D elementov ter terminskega plana je najbolj zamuden proces pri uporabi programa za izdelavo simulacij gradnje. Postopek povezovanja je sledeč:

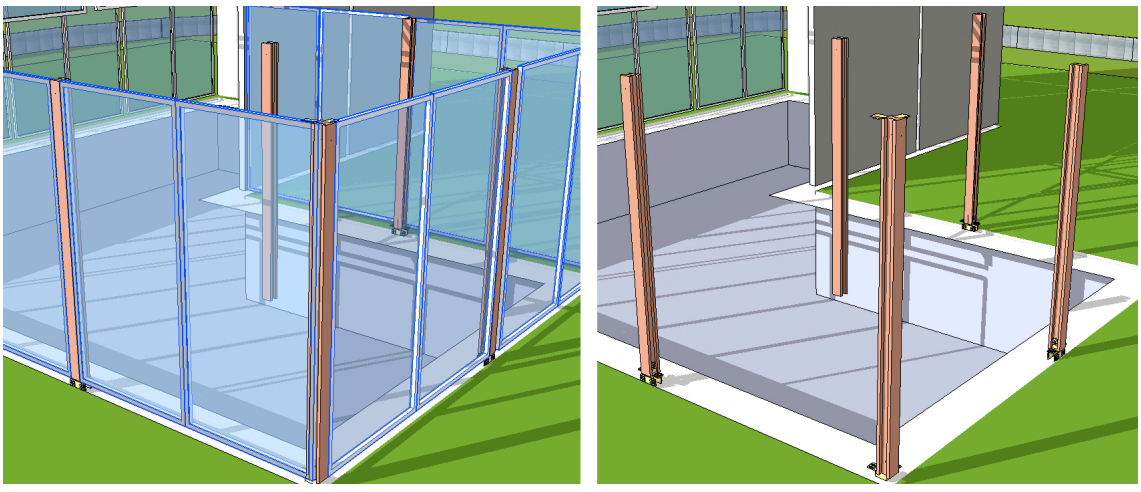
1. izberemo aktivnost v terminskem planu;
2. izberemo 3D elemente, katere želimo z izbrano aktivnostjo povezati;
3. pritisnemo gumb “Poveži izbrane elemente s terminskim planom”.

Če je povezovanje terminskega plana ter 3D elementov uspešno, se povezani elementi premaknejo v sloj, imenovan po imenu aktivnosti. Sloji, ki jih ustvari program, so nevidni, zato z zaslona izginejo vsi uspešno povezani elementi. Slika 30 prikazuje, kako okna pritličja, po povezovanju s terminskim planom izginejo z zaslona.

To, da uspešno povezani elementi izginejo z zaslona, olajša delo pri povezovanju 3D modela s terminskim planom, saj ostanejo vidni le še nepovezani 3D elementi. Seveda, lahko tekom povezovanja, vse že povezane elemente na zaslonu prikažmo in sicer prek podmenija Window—Layers, kjer sloje prikažemo, oziroma skrijemo.



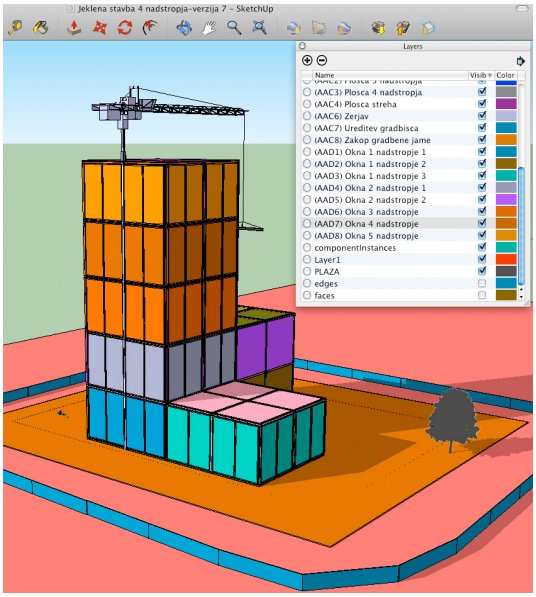
Slika 29: Izbrani 3D elementi (levo) in izbrana aktivnost v terminskem planu (desno)



Slika 30: Po uspešnem povezovanju 3D elementi postanejo nevidni
okna vidna pred povezavo (levo) in okna nevidna po povezavi (desno)

Poudariti je potrebno še eno lastnost programa Sketchup, ki nam koristi pri vizualizaciji povezanih elementov, in sicer prikaz modela tako, da so vsi elementi v 3D modelu pobarvani z različnimi barvami, glede na to kateremu sloju pripadajo. Prikaz lahko vklopimo ali izklopimo znotraj uporabniškega vmesnika s pritiskom na gumb "Pobarvaj po sloju/aktivnosti".

Na sliki 31 je 3D model, uspešno povezan s terminskim planom z vklopljenim barvanjem po slojih oziroma aktivnostih.



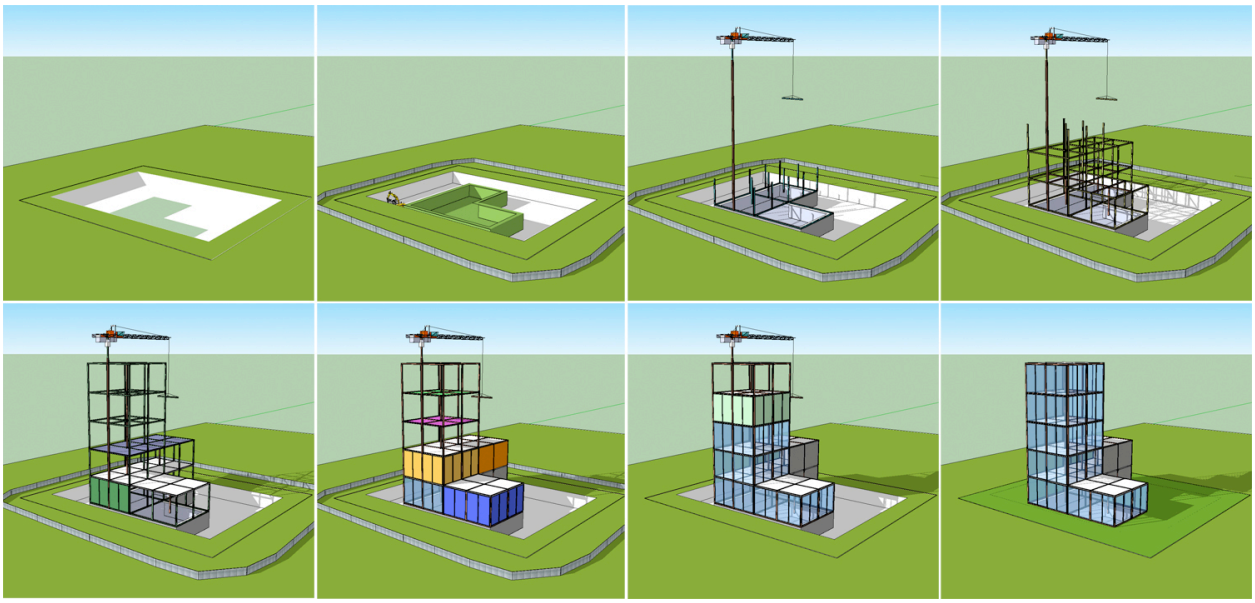
Slika 31: Povezani 3D model, z vklopljenim barvanjem po sloju oziroma aktivnosti

Po zaključenem povezovanju 3D modela ter terminskega plana, nadaljujemo s klikom na gumb "Naslednji korak", v zgornjem delu programskega okna za simulacije gradnje. Odpre se novo okno, kjer je potrebno izbrati trajanje simulacije s premikom drsnika med 5 in 120 sekund.

Po izboru trajanja simulacije ponovno s klikom na "Naslednji korak" preidemo na zadnjo fazo programa. Vsi sloji v 3D modelu postanejo nevidni, s pritiskom na ikono za zagon se simulacija prične. Med simulacijo se na ekranu izpisuje trenutni datum v terminskem planu. V kolikor so bili vpisani podatki o stroških posameznih aktivnosti, se prikaže tudi graf stroškov. V spodnjem delu

Slika 32: Nastavitev trajanja simulacije

se v tabeli prikazujejo vse trenutno izvajane aktivnosti. Za vse aktivnosti, katere so bile povezane s 3D modelom, se prikaže tudi barva, s katero so v 3D modelu med izvajanjem simulacije prikazani. Uporabnik v tabeli lahko prikaže le aktivnosti terminskega plana katere smo uspešno povezali s 3D modelom s pritiskom na "Prikaži le aktivnosti ki se nahajajo v 3D modelu".



Slika 33: Simulacija gradnje 4 nadstropnega objekta

7 ZAKLJUČEK

Izdelava simulacije gradnje objektov povezuje terminski plan s 3D modelom. V sklopu diplomske naloge smo želeli ponuditi izdelavo simulacije gradnje brez uporabe programskih paketov, ki simulacijo gradnje ponujajo vendar so zaradi cene in kompleksnosti nedostopni mnogim. Odločili smo se razviti program za terminske plane ter za izdelavo integriranih modelov in simulacij gradnje.

Program za terminske plane je dostopen vsakomur preko interneta, brezplačen ter za uporabo enostaven, izdelan je kot spletna aplikacija, ki teče na različnih računalniških sistemih, na katerih je naložen spletni brskalnik. V programu smo implementirali nekaj inovativnih rešitev. Program med izdelavi terminskega plana beleži vse spremembe aktivnosti, s tem sestavimo zgodovino uporabe programa in uporabniku omogočimo pregled zgodovine na nivoju terminskega plana ali aktivnosti. V terminski plan je mogoče dodati tudi stroške za vsako aktivnost ter prikazati porazdelitve stroškov in prispevkov po aktivnostih glede na čas.

Integrirani modeli so izdelani znotraj programa Sketchup, ki je brezplačen in omogoča pisanje razširitev s Sketchup Ruby API-jem. Sketchup je namenjen 3D modeliranju in je enostaven za uporabo, kljub temu pa je zelo zmogljiv. Izdelava integriranih modelov je mogoča s povezavo 3D elementov v Sketchup-u z aktivnostmi v terminskem planu, torej gre za interoperabilnost dveh programov. Z integriranim modelom je nadalje mogoča simulacija gradnje, saj zaradi povezave elementov ter terminskega plana poznamo za vsak 3D element datum začetka in zaključka gradnje. S tem je omogočena izdelava 4D simulacija gradnje, v kolikor pa so v terminski plan vpisani še stroški aktivnosti, pa je mogoče izdelati 5D simulacijo gradnje, ki prikaže tudi potek porabe stroškov med gradnjo.

Z obema programoma je omogočeno, da z enostavnimi in prosto dostopnimi orodji uporabniki sami izdelavijo integrirane modele in 4D ter 5D simulacije gradnje.

Pri izdelavi diplomske naloge in obeh programov smo spoznali, da je za opravilo določene naloge izjemno pomemben primeren izbor orodij za opravilo določene naloge. V našem primeru je imel Sketchup možnost uporabe spletne aplikacije kot uporabniškega vmesnika. Integracija obeh programov je bila tako veliko lažja, saj je bil tudi program za terminske plane izdelan kot spletne aplikacija, uporaba enakih podatkov za dve spletni aplikaciji pa je preprosta.

VIRI

Hardn, B. 2009. BIM and Construction Management, Proven tools, methods and workflows. Indianapolis. Wiley: str. 13, 14, 89, 90.

Estman, C., Teicholz, P., Sacks, R., Liston, K. 2008. BIM Handbook, A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors. Hoboken. John Wiley & Sons: 490 str.

Smith, D., Tardif, M. 2009. Building Information Modeling: A Strategic Implementation Guide. Hoboken. John Wiley & Sons: 216 str.

Krygiel, E., Nies, B. 2008. Green BIM. Indianapolis. Wiley: 337 str.

Rodošek, E. 1985. Operativno Planiranje. Ljubljana. VTOZD Oddelek za gradbeništvo in Oddelek za geodezijo FAGG, Univerza Edvarda Kardelja: 237 str.

Chonoles, M., Schardt J. 2003. UML 2 For Dummies. New York. Wiley: 412 str.

Ullman, L. 2009. Visual Quickstart Guide Ruby. Peachpit Press: 415 str.

Moock, C. 2007. Essential ActionScript 3.0. Sebastopol. O'Reilly Media: 911 str.

Lott, J., Patterson, D. 2007. Advanced ActionScript 3 with Design Patterns. Berkley. Peachpit Press: str. 3-32, 45-59, 97-101

Jackson, B. 2004. Construction Management JumpStart. Indianapolis. Wiley Publishing: 328 str.

Slovar informatike
http://www.islovar.org/iskanje_enostavno.asp (10.5.2010)

Adobe Flex Component Explorer
http://examples.adobe.com/flex3/componentexplorer/explorer.html (16.10.2009)

Flex 2 Style Explorer

http://examples.adobe.com/flex3/consulting/styleexplorer/Flex3StyleExplorer.html (16.10.2009)

Engineering ToolBox – Sketchup Edition

http://sketchup.engineeringtoolbox.com/ (8.3.2010)

Vertical alignment of inline images in LaTeX

http://johnjcamilleri.com/2010/05/vertical-alignment-of-inline-images-in-latex/ (27.5.2010)

Ruby Extended Icon List

http://dryicons.com/free-icons/icons-list/ruby-extended/ (9.11.2009)

Directed acyclic graph

http://wapedia.mobi/en/Directed_acyclic_graph (1.11.2009)

Coding Conventions – Flex SDK – Adobe Open Source

http://opensource.adobe.com/wiki/display/flexsdk/Coding+Conventions (13.9.2009)

Determining when an ArrayCollection changes in Flex

http://bit.ly/when-an-arraycollection-changes (29.10.2009)

Flex Date and Time (datetime) Picker Control

http://joelhooks.com/2008/10/11/flex-date-and-time-datetime-picker-control/ (29.10.2009)

Setting selectable ranges in Flex DateField control

http://bit.ly/setting-selectable-ranges-in-flex (13.11.2009)

Programming Ruby: The Pragmatic Programmer's Guide

http://www.ruby-doc.org/docs/ProgrammingRuby/html/tut_modules.html (4.3.2010)

Google Sketchup's Ruby–Javascript Interface

http://www.martinrinehart.com/models/rubies/ruby2javascript_javascript2ruby.html (14.3.2010)

RDoc Documentation

http://ruby-doc.org/core/ (11.2.2010)

Sketchup Ruby API Documentation

http://code.google.com/apis/sketchup/docs/index.html (25.1.2010)

Documentation – Google SketchUp Ruby Plugins Development

http://groups.google.com/group/SketchUp-Plugins-Dev (4.4.2010)

Google SketchUp API Blog: WebDialog

http://sketchupapi.blogspot.com/2007/12/webdialog.html (8.4.2010)

Beginning Ruby 2 – Write a Script

http://www.sketchucation.com/beginning-ruby-2-writing-a-script/ (10.1.2010)

Ruby Programming Language

http://www.ruby-lang.org/en/ (15.10.2009)

Ruby User's Guide

http://www.rubyist.net/ slagell/ruby/ (23.10.2009)

KAZALO PRILOG

PRILOGA A: FUNKCIJA POPULATEDATAFORANIMATION

```
private function populateDataForAnimation():void{
  //here let's start with empty array, that has nrOfFrames of empty fields;
  //the same with moneyAC;
  animationArr = [];
  moneyAC = new ArrayCollection();
  for (var j:int=0; j<nrOfFrames; j++){
        var datumStr:String = getDatum(new Date(startDate.fullYear, startDate.month,
            startDate.date + Math.round(nrDaysPerFrame)*j));
        moneyAC.addItem({date: j, money: 0, totalmoney:0, datum: datumStr, frame: j});
        animationArr.push('');
  }

  animationArr.push('');//se enega vec rabimo
  var endDate:String = getDatum(new Date(Application.application.finishDateForGantt));
  moneyAC.addItem({date: nrOfFrames, money: 0, datum: endDate, frame: nrOfFrames});
  var tempCollection:ArrayCollection = new ArrayCollection();
  var length:int = layersArr.length;
  for (var i:int=0; i<length; i++){
        var index:Number = layersArr[i]-1;
        var start:Number = dataForGantt.getItemAt(index).start;
        var duration:Number = dataForGantt.getItemAt(index).duration;
        var finish:Number = start+duration;
        var newStart:int = Math.round(start/nrDaysPerFrame);
        var newFinish:int = Math.round(finish/nrDaysPerFrame);
        if (newStart == newFinish && newStart != nrOfFrames){newFinish +=1;}//nocemo da se
            zgodi, da se v istem frejmu prikaze pobarvan element, in tudi navadno prikazan.
        //tempCollection.addItem({s: newStart, f:newFinish, guid: dataForGantt.getItemAt(
            index).guid, taskname: dataForGantt.getItemAt(index).taskname,
        //color: dataForGantt.getItemAt(index).color, myuid: dataForGantt.getItemAt(index).
            myuid});
        var layerName:String = '(' + dataForGantt.getItemAt(index).guid + ')' + ' ' +
            dataForGantt.getItemAt(index).taskname;
        var red:Number = (dataForGantt.getItemAt(index).color >> 16) & 0xff;
        var green:Number = (dataForGantt.getItemAt(index).color >> 8) & 0xff;
        var blue:Number = (dataForGantt.getItemAt(index).color & 0xff);
        var RGB:String = red.toString() + ',' + green.toString() + ',' + blue.toString();
        if (newStart != nrOfFrames){
```

```
            //tu preprecimo, da se start pojavi kot zadnji frame, le finish je lahko zadnji
            tempCollection.addItem({frame: newStart, type:'start', layer_name: layerName,
                color: RGB, uid: index});                }
            tempCollection.addItem({frame: newFinish, type:'finish', layer_name: layerName,
                color: '', uid: index});
    }

    for (var k:int=0; k<tempCollection.length; k++){
           var frame:int = tempCollection.getItemAt(k).frame;
           animationArr[frame] += tempCollection.getItemAt(k).type + '--||--' + tempCollection.
               getItemAt(k).layer_name + '--||--' + tempCollection.getItemAt(k).color +
               '--||--';
    }
    generateMoneyData();
    //naredimo se vse potrebno za tasksAC
    generateTasksAC();
    //tukaj imamo animationARR in place. in gremo lahko naprej z animacijo.
    animationTimer = new Timer(timerStep*1000, nrOfFrames);
    animationTimer.addEventListener(TimerEvent.TIMER, frameEnter);
    animationTimer.addEventListener(TimerEvent.TIMER_COMPLETE, lastFrame);
    //stage.addEventListener(TimerEvent.TIMER, frameEnter);
}
```

PRILOGA B: FUNKCIJA ANIMATIONCALL

```
dialog.add_action_callback("animationCall"){|dialog, p|
  puts 'animation Call made with p: ' + p
  dataForAnimation = p.split('--||--')
  length = dataForAnimation.length
  (0...length).step(3){|i|
    layerName = dataForAnimation[i+1]
    sf = dataForAnimation[i]
    color = ''
    if (sf != 'finish')
      color = dataForAnimation[i+2].split(',')
      color = [color[0].to_i, color[1].to_i, color[2].to_i]
    end
    entities = model.entities
    current_layer = ''
    if check_layer(layerName){|layer| current_layer = layer}
      if (sf == 'start')
        #tukaj naredimo potrebno ce je start
        current_layer.visible = true
        #loop cez vse entitie
        entities.each {|ent|
          if (ent.layer.name == layerName)
            if (ent.get_attribute("onTimeData", "hideAtEnd" , '') != true)
              color_entity(ent, color)
            end
          end
        }
      else
        #tukaj naredimo potrebno ce je finish
        current_layer.visible = true
        #loop cez vse entitite
        entities.each {|ent|
          if (ent.layer.name == layerName)
            if (ent.get_attribute("onTimeData", "hideAtEnd" , '') != true)
              bring_back_old_material(ent)
            else
              ent.visible = false
            end
```

```
          end
        }
      end
    else
      puts('problem in finding layer named: ' + layerName)
    end
  }
}
```

PRILOGA C: FUNKCIJA WBSSWITCH

```
private function WBSSwitch(e:MouseEvent):void{
  //iz eventa e dobimo kateri gumb je bil kliknjen
  var gumb:String = e.currentTarget.id;
  //preverimo, ce je izbrana vrstica v Datagridu, ce vrstica ni oznacena obvestimo
      uporabnika
  if (dataGridGantt.selectedItem == null){
    //prikaze sporocilo in zazenem timer, cez 5 sekund zelimo da se error msg zbrise
    msgLabelTimer(selectRowInProjectStr, selectRowInProjectExplainStr);
  } else {
    var selectedIndex:Number = dataGridGantt.selectedIndex;
    var selectedItem:Object = dataForGantt[selectedIndex];
    //ce kliknemo prvega v tabeli, prvi task v tabeli mora biti glavni task, ne moremo nic
        niti z gumbom levo niti desno!
    //ce ni prvi v tabeli, gledamo vse ostalo
    if (selectedIndex == 0){
      msgLabelTimer(operationNotPossibleStr, firstTaskNotMovableExplainStr);
    } else {
      //preverimo ali je bil kliknjen levi ali desni gumb
      //zapisemo si outlinelevel izbranega elementa in WBS
      var currentOutlinelevel:Number = selectedItem.outlinelevel;
      var currentWBS:String = selectedItem.WBS;
      var tempCurrentWBS:String;
      var i:Number,j:Number;
      var exitOL:Number = 0; //spremeniljivka da lahko posebej spremenimo WBS in
          outlinelevel
      if (gumb == 'WBSleft'){
        //pritisnjen je gumb na levo stran
        //preverimo ali je outlinelevel 1, ce je enak 1, potem userju povemo, da opreacije
            ni mogoce izvesti
        if (currentOutlinelevel == 1){
          msgLabelTimer(operationNotPossibleStr, selectedItem.guid + ': ' +
              taskCannotGoLowerStr);
        } else {
          //shranimo WBS in OL data
          saveWBSandOLData();
          //pritisnjen je lefi gumb in operacija je mogoca...
          //spremenimo WBS elementa, ki je oznacen (od outlinelevela je odvnisno katero
```

```
            decimalko zamaknemo)
        selectedItem.outlinelevel -= 1;
        selectedItem.WBS = changeWBS(currentWBS, 'down' , selectedItem.outlinelevel);
        addHistory(selectedItem.guid, 'WBS', currentWBS, selectedItem.WBS);
        addHistory(selectedItem.guid, 'Outlinelevel', (selectedItem.outlinelevel+1).
            toString(), selectedItem.outlinelevel, true);
        //zdaj gremo spremenit se outlinelevele za vse ostale vnose, kjer je to potrebno!
        for (i=1; i<(dataForGantt.length - selectedIndex); i++){
          j = selectedIndex + i;//tekoci index
          if (currentOutlinelevel - 1 > dataForGantt[j].outlinelevel){ //ce je to res
              potem prekinemo z iskanjem, ker nam ni potrebno spremeniti niti WBS niti
              outlinelevel!
            i = dataForGantt.length; //gremo ven iz loopa!
          } else if(currentOutlinelevel == dataForGantt[j].outlinelevel ||
              currentOutlinelevel - 1 == dataForGantt[j].outlinelevel){
            tempCurrentWBS = dataForGantt[j].WBS;
            dataForGantt[j].WBS = changeWBS(dataForGantt[j].WBS, 'down',dataForGantt[j].
                outlinelevel, dataForGantt[j-1].WBS, i);
            addHistory(dataForGantt[j].guid, 'WBS', tempCurrentWBS, dataForGantt[j].WBS,
                true);
            //to damo zato, ker ce je eden izmed outlinelevelov enak ne zelimo sploh
                gledati naslednje in jih spremninjati
            //spremeniti pa jih moramo zaradi WBSa
            exitOL = 1;
          } else if (currentOutlinelevel < dataForGantt[j].outlinelevel){
            if (exitOL == 0) dataForGantt[j].outlinelevel--;
              tempCurrentWBS = dataForGantt[j].WBS;
              dataForGantt[j].WBS = changeWBS(dataForGantt[j].WBS, 'down',dataForGantt[j].
                  outlinelevel, dataForGantt[j-1].WBS, i);
              addHistory(dataForGantt[j].guid, 'WBS', tempCurrentWBS, dataForGantt[j].WBS,
                   true);
              if (exitOL == 0) addHistory(dataForGantt[j].guid, 'Outlinelevel', (
                  dataForGantt[j].outlinelevel+1).toString(),dataForGantt[j].outlinelevel,
                   true);
          }
        }
        checkAllConnections();
      }
    } else {
```

```
//pritisnjen je gumb na desno
//preverimo kaksen outlinelevel ima zgornji vnos v tabeli, ce je je outlinelevel
    manjsi od izbranega opracije ni mogoce opraviti
if (currentOutlinelevel > dataForGantt[selectedIndex-1].outlinelevel){
  msgLabelTimer(operationNotPossibleStr, selectedItem.guid + ': ' +
      taskCannotGoHigherStr);
} else {
  //shranimo WBS in OL data
  saveWBSandOLData();
  //pritisnjen je desni gumb in operacija je mogoca...
  //dobimo outlinelevel in ga povecamo za 1
  selectedItem.outlinelevel += 1;
  selectedItem.WBS = changeWBS(currentWBS, 'up' , selectedItem.outlinelevel,
      dataForGantt[selectedIndex-1].WBS);
  addHistory(selectedItem.guid, 'WBS', currentWBS, selectedItem.WBS);
  addHistory(selectedItem.guid, 'Outlinelevel', (selectedItem.outlinelevel-1).toString
      (), selectedItem.outlinelevel, true);
  //pregledamo se naslednje vnose, ce imajo outline level vecni kot
      currentoutlinelevel, pomeni,
  //da so zapisani pod ta vnos in moramo tudi njihov outline povecati
  for (i=1; i<(dataForGantt.length - selectedIndex); i++){
    j = selectedIndex + i;//tekoci index
    if (currentOutlinelevel == dataForGantt[j].outlinelevel){
      exitOL = 1;
      tempCurrentWBS = dataForGantt[j].WBS;
      dataForGantt[j].WBS = changeWBS(dataForGantt[j].WBS, 'up',dataForGantt[j].
          outlinelevel, dataForGantt[j-1].WBS, i);
      addHistory(dataForGantt[j].guid, 'WBS', tempCurrentWBS, dataForGantt[j].WBS,
          true);
    } else if (currentOutlinelevel < dataForGantt[j].outlinelevel){
      if (exitOL == 0) dataForGantt[j].outlinelevel++;
      tempCurrentWBS = dataForGantt[j].WBS;
      dataForGantt[j].WBS = changeWBS(dataForGantt[j].WBS, 'up',dataForGantt[j].
          outlinelevel, dataForGantt[j-1].WBS);
      addHistory(dataForGantt[j].guid, 'WBS', tempCurrentWBS, dataForGantt[j].WBS,
          true);
      if (exitOL == 0) addHistory(dataForGantt[j].guid, 'Outlinelevel', (dataForGantt[
          j].outlinelevel-1).toString(),dataForGantt[j].outlinelevel, true);
    } else {
```

```
            i = dataForGantt.length;
          }
        }
        checkAllConnections();
      }
    }
    dataForGantt.refresh();
    }
   }
}

private function changeWBS(curentWBS:String, direction:String, curentOL:Number, previousWBS:
    String = '', index:Number = 0):String{
  //curentWBS - podamo WBS katerega zelimo spremeniti
  //direction - up ali down
  //previousWBS - velja le za direction up, ker takrat ne moremo tocno dolociti prejsnji wbs
      ...
  var WBS:String = '';
  var WBSArr:Array = curentWBS.split('.');
  var previousWBSArr:Array;
  var outlinelevel:Number = curentOL; //ko razbijemo array iz dobljenega WBS-a tocno vemo
      kateri outline level je
  var previousOutlinelevel:Number;
  if (direction == 'up'){
    previousWBSArr = previousWBS.split('.');
    previousOutlinelevel = previousWBSArr.length;
    if (outlinelevel > previousOutlinelevel){//primer da je prejsni vecji od zdajsnega,
        pomeni da bosta po operaciji na istem outlinelevelu, zato prejsnjemu le dodamo .1
      WBS += previousWBS + '.1';
        } else if (outlinelevel == previousOutlinelevel){
          for (i=0; i<outlinelevel; i++){
            if (i!=0) WBS += '.';
            if (i !=outlinelevel-1){
              WBS += previousWBSArr[i];
                } else {
              previousWBSArr[i]++;
              WBS += previousWBSArr[i];
                }
        }
```

```
        } else {
      for (i=0; i<outlinelevel; i++){
        if (i!=0) WBS += '.';
        if (i != outlinelevel-1){
          WBS += previousWBSArr[i];
        } else {
          previousWBSArr[i]++;
          WBS += previousWBSArr[i];
        }
      }
        }
  } else {
    if (previousWBS == ''){ //ce WBS ni podan potem spremenimo WBS ne glede na predhodnji
        clen. (ponavadi uporabimo za tisti clen ki smo ga oznacili mu zmanjsali WBS za 1
      for (i=0; i<outlinelevel; i++){
        if (i!=0) WBS += '.';
        if (i !=outlinelevel-1){
          WBS += WBSArr[i];
        } else {
          WBSArr[i]++;
          WBS += WBSArr[i];
        }
      }
    } else {
      previousWBSArr = previousWBS.split('.');
      previousOutlinelevel = previousWBSArr.length;
      //ce sta previousOutlinelevel in outline level enaka potem spremenimo le zadnjo
          stevilko kot smo to naredili zgoraj, le da gledamo previousWBSArr in ne WBSArr
      if (previousOutlinelevel == outlinelevel && index !=1){
        for (i=0; i<outlinelevel; i++){
          if (i!=0) WBS += '.';
          if (i !=outlinelevel-1){
            WBS += previousWBSArr[i];
          } else {
            previousWBSArr[i]++;
            WBS += previousWBSArr[i];
          }
        }
      } else if (previousOutlinelevel > outlinelevel){
```

```
        var razlika:Number = previousOutlinelevel - (previousOutlinelevel - outlinelevel);
        for (i=0; i<razlika; i++){
          if (i!=0) WBS += '.';
          if (i !=razlika-1){
            WBS += previousWBSArr[i];
          } else {
            previousWBSArr[i]++;
            WBS += previousWBSArr[i];
          }
        }
      } else if (previousOutlinelevel == outlinelevel && index == 1){
        //primer ko recimo spremenimo task 1.2 v 2, naslednji vnos je prej imel 2 in ga tu
            pravilno naredi v 3, namesto 2.1
        for (i=0; i<outlinelevel; i++){
          if (i!=0) WBS += '.';
          if (i !=outlinelevel-1){
            WBS += WBSArr[i];
          } else {
            WBSArr[i]++;
            WBS += WBSArr[i];
          }
        }
      } else {
        WBS += previousWBS + '.1';
      }
    }
  }
  return WBS;
}
```

PRILOGA D: FUNKCIJA CONNECTION_DO_CHANGE

```
private function connection\_do\_change():void{
 //endDate = new Date(this.data.start_date.fullYear, this.data.start_date.month, (this.data.
     start_date.date + this.data.duration));//dobimo endDate za sedajsnji task!
 endDate = new Date('1/1/1980');//zgornjega uporabimo ce nocemo da se task premakne, ce
     endDate (sedanji) vecji od tistega ki bi ga dobili zaradi connectiona
 currentID = Number(this.data.uid);
 var connection:String = this.text;
 var error:Number = 0;
 if (StringUtil.trim(connection) == ""){//zbisali smo connection, zdej naredimo potrebno (ce
      kaj) da si to uredimo
  emptyConnection();
 } else {
  //razbijemo string od vejice
  var connectionsArray:Array = connection.split(',');//ce imamo 14, 16 FS, 17 - razbijemo v
      14 - 16 FS in 17.
  var connAC:ArrayCollection = new ArrayCollection;
  var connectionEnd:String = ''; //v to spremenljivko bomo vpisali koncni connection string,
       in s tem popravili kaksno napako ki jo je user napisal!
  var duplicates:Number=0;
  for (var i:uint=0; i<connectionsArray.length; i++){
   //zdaj pogledamo ce je v tempConn zapisan se lag, recimo 1 SS (3), ali 1 FS (4), zelimo
       dobiti 3 oz 4.
   var currentLag:Number = 0;
   var startIndex:Number = connectionsArray[i].toString().indexOf('(');
   if (startIndex != -1){
    var endIndex:Number = connectionsArray[i].toString().indexOf(')');
    if (endIndex != -1){
     currentLag = Number(connectionsArray[i].toString().slice(startIndex + 1, endIndex));
     if (currentLag.toString() != 'NaN'){
      //tukaj vemo da je currentLag stevilo...zapisali smo si ga v spr. currentLag in sedaj
          lahko lag zbrisemo iz povezave, da bo vse delovalo kot prej
      connectionsArray[i] = connectionsArray[i].toString().replace('('+ currentLag + ')',
          '');
     }
    }
   }
   connectionsArray[i] = StringUtil.trim(connectionsArray[i]);
```

```
//razbijemo vsak vnos, da vidimo ce je vpisan FF ali FS, vnos ne sme imeti vec kot 2 dela
     (ali je samo cifra ali cifra in tip FF ali FS)
var tempConn:Array = connectionsArray[i].split(' ');
var connType:String = '';
if (tempConn.length > 2) {
 error = 1; i = connectionsArray.length; trace('error 1 was called: too many arguments');
      //klicemo napako! prevec argumentov, ce napisemo 14 FS 15 (brez vejice)
} else {
 if (isNaN(tempConn[0]) || tempConn[0] == "") {
  error = 2; i = connectionsArray.length; trace('error 2 was called: Not Number'); //prvi
       argument ni stevilo
 } else {
  if(tempConn.length == 2){
   if(tempConn[1] == 'FF') {connType = 'FF';}
   else if (tempConn[1] == 'FS') {connType = 'FS';}
   else if (tempConn[1] == 'SS') {connType = 'SS';}
  else {
  //ce ni bil kot drugi argument vpisan FF ali FS, potem zapisemo notri kar FS...(primer
      pisemo 1, 2 FFF, 3 spremeni v 1, 2 FS, 3-
  this.data.connections = this.data.connections.replace(tempConn[1], 'FS');
  connType = 'FS';
 }
}
//pregledamo duplikate!
for (var j:uint=0; j<connAC.length; j++){
 if (tempConn[0] == connAC.getItemAt(j).connID){
  trace('duplicate');
  j=connAC.length;//skocimo ven iz zanke!
  duplicates = 1;
 } else {
  trace('not Duplicate');
 }
}
//Ce je duplikat - ga vzemo vn!//
if (duplicates == 1){//zadnja stevlika ki smo jo gledali je duplikat, zato je ne bomo
    vpisali v prenosni connAC in izbrisali iz polja connections
 //prikazemo error (ne uporabimo error msg ker nas error funckija vrne v prejsnje stanje,
      kar ne zelimo!
 document.msgLabelTimer(duplicatesFoundStr);
```

```
} else {//ni duplikatov!
 //Tukaj preverimo ali je connection dovoljen ali ne!//
 trace('Check for connection: ' + tempConn[0]);
 if (checkConnection(tempConn[0])){
  var endDateOfCurrentConn:Date;
  //vse potrebno imamo in lahko zapisemo v arraycollection connAC
  connAC.addItem({id: this.data.uid,connID: tempConn[0], connType: connType, connLag:
      currentLag});
  if (connType == ''){
   //connectionEnd += tempConn[0] + ', ';
   connectionEnd += tempConn[0] + ' FS (' + currentLag + '), ';
  } else {
   connectionEnd += tempConn[0] + ' ' + connType + ' (' + currentLag + '), ';
  }
  duplicates = 0;
  //zapisemo si od connectiona, ki smo ga pregledali in je OK Finish Date!
  var currentStartDate:Date = new Date(document.dataForGantt.getItemAt(tempConn[0]-1).
      start_date);
  endDateOfCurrentConn = new Date(currentStartDate.fullYear, currentStartDate.month, (
      currentStartDate.date + document.dataForGantt.getItemAt(tempConn[0]-1).duration));
  var endDateWithCurrentConn:Date; //sem si spravimo kdaj bi bil task koncan, ce bi bil
      task ki ga zdaj pregledamo odvisen samo od current connection
  if (connType == 'FF'){
   endDateWithCurrentConn = endDateOfCurrentConn; //ker je FF se current task konca
       takrat kot connection
   endDateType = 'FF';
  } else if (connType == 'SS'){
   endDateWithCurrentConn = new Date(currentStartDate.fullYear, currentStartDate.month, (
       currentStartDate.date + this.data.duration));
   endDateType = 'SS';
  } else {
   endDateWithCurrentConn = new Date(endDateOfCurrentConn.fullYear, endDateOfCurrentConn.
       month, (endDateOfCurrentConn.date + this.data.duration));
   endDateType = 'FS';
  }
  //ce je trenutni Finish Date vecji od tiostega ki je v spremenljivki endDate, ga
      zamenjamo
  if (endDateWithCurrentConn > endDate){
   endDate = endDateWithCurrentConn;
```

```
    }
   } else {
    trace('error in one connection');
   }
  }//end ni duplokatov
 }
 }
 }
 //Zapisemo v connection polje!//
 connectionEnd = connectionEnd.slice(0, -2);
 this.data.connections = connectionEnd;
 if (connectionEnd != ""){
  document.addHistory(this.data.guid, 'connections',connection_beginning,connectionEnd);
  //poklicemo funkcijo, ki bo nove povezave vpisala v connectionsAC
  document.addValuesToConnectionsAC(connAC, this.data.uid);
  //v spremenlivki endDate imamo najkasnejsi datum, ki je odvisen od povezav (lahko se
      seveda zgodi, da ima task svoj lastni datum vecji od zadnjih datumov connectiona.
  //v spremenljivki endDateType imamo pa zapisan tip povezave. Ce je tip povezave enak '',
      potem vemo da se datum ni spremenil (nobeden izmed connectionov ni imel kasnejsega
      datuma)
  //in potrebno ni narediti nic. Cene glede na endDateType dolocimo zacetek in konec taska.
  //pregledamo se lag in dodamo lag ce ni enak 0;
  var startDate:Date;
  if (currentLag != 0){
   startDate = new Date(endDate.fullYear, endDate.month, (endDate.date - this.data.duration
       + currentLag));
  } else {
   startDate = new Date(endDate.fullYear, endDate.month, (endDate.date - this.data.duration
      ));
  }
  //recimo da imam taske A, B, C in D. Task B je oce Cja. Najprej povezemo B z A. B in C
      skocita na konec taska A
  //kaj se zgodi ce sedaj task C povezemo s taskom D (ki ima konec prej kot konec A). Task
      B bo skocil nazaj, ceprav ne sme ker je njegov oce (B)
  //povezan z A in se zacne kasneje. S tem delom to preprecimo!
  hierarchyData = document.hierarchyData;
  //shranimo si se connectionsAC
  var connectionsAC:ArrayCollection = document.getConnectionsAC();
  //najprej loop cez hierarhijo
```

```
  for (var k:int=0; k<hierarchyData.length; k++){
   if (hierarchyData.getItemAt(k).uid == this.data.uid){
    for (var l:int=0; l<connectionsAC.length; l++){
     if (hierarchyData.getItemAt(k).dep == connectionsAC.getItemAt(l).id){
      var checkStartDate:Date = document.dataForGantt.getItemAt(hierarchyData.getItemAt(k).
          dep).start_date;
      if (checkStartDate > startDate){
       startDate = checkStartDate;
      }
     }
    }
   }
  }
  //spremenimo start date ce je pot
  var oldStartDate:Date = this.data.start_date;
  if (oldStartDate != startDate){ //naredi vse to samo ce se je kaj spremenilo
   document.addHistory(this.data.guid, 'startDate',oldStartDate,startDate.toString(), true)
       ;
   this.data.start_date = startDate;
   document.checkHierarchyForStartFinishChanges(this.data.uid-1, oldStartDate); //poklicemo
        funckijo ki opravi potrebno da se vsi taski pravilno postavijo!
   //spremenimo StartRange za ta task - ampak samo v primeru da gre za povezavo FS, ce gre
       za povezavo FF potem nismo vezani na zacetek taska
   if (endDateType == 'FS'){
    this.data.startRange = startDate;
   }
   //naredi potrebno za update Gantovega diagrama//
   document.updateDateForGantt(startDate, endDate);
   //spremeni start v ArrayCollectionu, da bo pravilno prikazan
   this.data.start = document.durationBetweenDates(startDate, document.startDateForGantt);
  }
 }
 }//end of if(connectionEnd !=0)
 if (error != 0){
  errorCall(error);
 }
 //preglej ali je connection mogoc.
 trace('id: ' + currentID + ' - connection: ' + connection);
}
```

PRILOGA E: FUNKCIJA CONNECTIONCHECK

```
private function checkConnection(connection:Number):Boolean {
 var i:Number;
 var connectionData:ArrayCollection = new ArrayCollection;
 var hierarchyForConnection:Array = []; //v ta array si shranimo hierarhijo od connectiona,
     ki ga pregledujemo
 var connectionsAC:ArrayCollection = document.getConnectionsAC();
 currentIDDep = [];
 currentIDisDep = [];
 hierarchyData = document.hierarchyData;
 //preveri ce je link sam nase!
 if (connection == currentID){
  trace('error in coonection: Link to it self');
  document.msgLabelTimer(errorLinkToItsefStr);
  return false;
 }
 //preveri ali sploh obstaja vnos z id-jem linka oz. connectiona//
 var numberOfTasks:Number = document.dataForGantt.length;
 if (numberOfTasks < connection){
  trace('error in connection: Connection is bigger than the id of the last task');
  document.msgLabelTimer(errorIdTooBigStr);
  return false;
 }
 //preveri ce je vse OK z hierarhijo//
 hierarchyForId.push(currentID);//doda med triggerje samega sebe!
 for (i=0; i<hierarchyData.length; i++){
  //v tej vrtici si zapisemo vse dependencies za id, kar bomo pri naslednjem odseku
      potrebovali - to bodo nasi 'triggerji'
  if (hierarchyData.getItemAt(i).uid == currentID){hierarchyForId.push(hierarchyData.
      getItemAt(i).dep); currentIDDep.push(hierarchyData.getItemAt(i).dep);}
  if (hierarchyData.getItemAt(i).dep == currentID){hierarchyForId.push(hierarchyData.
      getItemAt(i).uid); currentIDisDep.push(hierarchyData.getItemAt(i).uid);}
  //tulaj pregledamo hierarhijo tudi za connection ki ga pregledujemo, da dobimo se
      hierarhijo zanj!
  if (hierarchyData.getItemAt(i).uid == connection){hierarchyForConnection.push(
      hierarchyData.getItemAt(i).dep);}
  if (hierarchyData.getItemAt(i).dep == connection){hierarchyForConnection.push(
      hierarchyData.getItemAt(i).uid);}
```

```
 if (hierarchyData.getItemAt(i).uid == currentID && hierarchyData.getItemAt(i).dep ==
     connection){
 document.msgLabelTimer(errorHierarchyStr);
 trace('error in connection: Hierarchy Error');
 return false;
}
}
//preverimo ali ni slucajno connection ze linkan na current id//
//v hierarchyForId imamo t.i triggerje, zdej je potrebno zapolniti arrayForTrriggers, da
    vidimio ce je connection mogoc ali ne
var allConnections:String = '';
var currentConnection:String = StringUtil.trim(document.dataForGantt.getItemAt(connection
    -1).connections);
//dodamo connection, ki je vpisano v IDju connectiona ki ga pregledujemo
if (currentConnection != '') {allConnections += currentConnection + ',';}
 for (i=0; i<hierarchyForConnection.length; i++){
  currentConnection = StringUtil.trim(document.dataForGantt.getItemAt(
      hierarchyForConnection[i]-1).connections);
  if (currentConnection != '') {allConnections += currentConnection + ',';}
 }
 if (allConnections != ''){
  //ker ima zadnji vnos v allconnections na koncu vejico, jo s spodnjim ukazom odstranimo!
  allConnections = allConnections.slice(0, -1);
  //v allConnections je recimo zapis: 1,3,4,6,10 FF, 12 FF
  var allConnectionsArray:Array = [];
  allConnectionsArray = allConnections.split(',');
  for (i=0; i<allConnectionsArray.length; i++){//iz allConnectionsArray vzamemo samo
      connection ven, torej zanemarimo tip povezave (FF ali FS)
   allConnectionsArray[i] = StringUtil.trim(allConnectionsArray[i]);
   var justConnection:Array = allConnectionsArray[i].split(' ');
   arrayForTriggers.push(justConnection[0]);//zapisemo vse connectione ki nas zanimajo v
       arrayForTriggers
  }
 }
 //sedaj imamo v arrayForTriggers vse povezave, ki jih imajo elementi za connection ki ga
     preverjamo (ce preverjamo connection=2 in ima 2 otroke 3 in 4, smo
 //pregledali vse conncetione ki so v 2, 3, 4. v arrayForTriggers se pojavijo connectioni
     od 2, 3, 4.
 //sedaj moramo pa za te konectione ki smo jih nasli se enkrat preveriti ali imajo oni spet
```

```
        kake connectione in jih dodati v arrayForTriggers
    for (i=0; i<arrayForTriggers.length; i++){
     var tempHierarchyForConnection:Array = [arrayForTriggers[i]];
     for (j=0; j<hierarchyData.length; j++){//dobimo dodatni hierarchy data za connection ki
         ga pregledamo
      if (hierarchyData.getItemAt(j).uid == arrayForTriggers[i]){tempHierarchyForConnection.
          push(hierarchyData.getItemAt(j).dep);}
      if (hierarchyData.getItemAt(j).dep == arrayForTriggers[i]){tempHierarchyForConnection.
          push(hierarchyData.getItemAt(j).uid);}
     }
     for (j=0; j<tempHierarchyForConnection.length; j++){
      for (var k:int=0; k<connectionsAC.length; k++){
       if (connectionsAC.getItemAt(k).id == tempHierarchyForConnection[j]){
        arrayForTriggers.push(connectionsAC.getItemAt(k).connID);
       }
      }
     }
    }
    //na koncu moramo se za vsak element iz hierarchyForId - ki je za nas t.i. trigger
        preveriti ali imamo v arrayForTriggers enak vnos, ce ga imamo je error v povezovanju!
    for (i=0; i<hierarchyForId.length; i++){
     var trigger:Number = hierarchyForId[i];
     for (var j:uint=0; j<arrayForTriggers.length; j++){
      if (trigger == arrayForTriggers[j]){
       //document.tukaj.text = 'Trigger se je vklopil za id: ' + trigger;
       trace('Trigger went off for ID: ' + trigger + ' - We get circular connection');
       document.msgLabelTimer(errorCircularConnectionStr);
       return false;
      }
     }
    }
    //ce gre vse skozi vrne pozitiven rezultat:)
    return true;
   }
```