TEHNIČKA ŠKOLA RUĐERA BOŠKOVIĆA  
GETALDIĆEVA 4, ZAGREB

ZAVRŠNI STRUČNI RAD:

CreateUI

MENTOR: Goran Gomaz UČENIK: Roko Burilo  
 RAZRED: 4.H

Zagreb, svibanj 2020.

Sadržaj

[I. Uvod 2](#_Toc40803584)

[II. CUI framework 4](#_Toc40803585)

[A. Struktura CUI sustava 4](#_Toc40803586)

[B. Styles podsustav 5](#_Toc40803587)

[a) Svrha 6](#_Toc40803588)

[b) Format 6](#_Toc40803589)

[c) Parsiranje formata 7](#_Toc40803590)

[d) Primjeri datoteka 8](#_Toc40803591)

[e) Schematic klasa 9](#_Toc40803592)

[f) Node klasa 9](#_Toc40803593)

[C. Scenes podsustav 10](#_Toc40803594)

[a) Svrha 10](#_Toc40803595)

[b) Format 10](#_Toc40803596)

[c) Parsiranje formata 10](#_Toc40803597)

[d) Primjeri datoteka 11](#_Toc40803598)

[e) SceneGraph 12](#_Toc40803599)

[D. CUI-SFML binding 13](#_Toc40803600)

[a) Window klasa 13](#_Toc40803601)

[b) RenderCache 14](#_Toc40803602)

[c) EventData 15](#_Toc40803603)

[d) TimerEvent 15](#_Toc40803604)

[e) VisualElement 16](#_Toc40803605)

[f) WindowOptions 16](#_Toc40803606)

[g) Templates 16](#_Toc40803607)

[h) Intermediaries 16](#_Toc40803608)

[E. Primjeri aplikacija 17](#_Toc40803609)

[a) Gumb koji mijenja boju pozadine 17](#_Toc40803610)

[b) Više gumbova koji djeluju na isti blok 22](#_Toc40803611)

[c) Tipkovnica 28](#_Toc40803612)

[F. Zaključak 34](#_Toc40803613)

1. Uvod

CreateUI je završni rad softverskog tipa čiji je cilj osmisliti i implementirati framework za stvaranje i upravljanjem GUI sustava. Razlog odabiru ovog rada je bilo u stjecanju prakse u izradi velikih softverskih projekata. Različiti manji dijelovi ovog projekta mogu se gledati i samostalno jer su i to projekti vrijedni spomena.

Tijekom izrade ovog projekta izradio se:

* Svojevrsni jezik koji opisuje strukturu scena
* Svojevrsni jezik koji opisuje stilove GUI elemenata (Block-ova)
* Parser[[1]](#footnote-1) za ta dva jezika
* Dizajn sučelja između CUI-ja i različitih *binding-a[[2]](#footnote-2)*
* Binding CUI-ja sa multimedijskom bibliotekom SFML zvan CUI-SFML
* Nekoliko malih primjera mogućih GUI programa u CUI-SFML biblioteci
* Parsiranje tijekom kompilacije i primitivan oblik rukovanje parsing greškama

U projektu se koristio C++17 zbog olakšavanja restrikcija na *constexpr* uvjetima rada, Tessilova implementacija rječnika zvan tsl::hopscotch\_map, SFML zbog lakog ugrađivanja biblioteke i jednostavnog sučelja prema grafici.

Obje biblioteke (CUI i CUI-SFML) su oblika *header-only* zbog lakog prototipiranja i ugrađivanja u druge projekte.

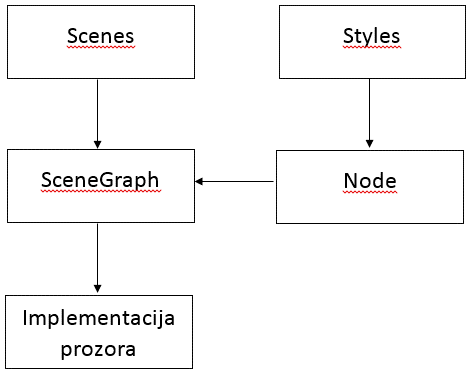
1. CUI framework
2. Struktura CUI sustava

Strukturu CUI sustava čine podsustavi Styles i Scenes.

Styles određuje stilove čije atribute definira korisnik, tj. pisac stila. Ti stilovi se onda mogu iskoristiti u Scenes sustavu, gdje jedan GUI blok može definirati stilove koje će koristiti.

Scenes sustav određuje samu strukturu, tj. odnos članova scenskog grafa. On definira pojedine značajke svakog člana, kao što su stilovi i tekst.

Ta dva sustava spaja SceneGraph klasa skupa sa Node klasom. Node klasa sadrži atribute i pravila, dok SceneGraph sadržava Node-ove.



Slika 1. Shema odnosa podsustava unutar CUI sustava.

1. Styles podsustav

Styles je podsustav CUI-a i format koji definira stilove koje onda može Scenes podsustav iskoristiti za svoje blokove.

Mogući atributi su:

* background – postavlja pozadinu na boju ili sliku; Moguće vrijednosti:
  + rgb(crvena, zelena, plava)
  + rgba(crvena, zelena, plava, alpha)
  + url(put\_to\_datoteke)
  + red
  + green
  + blue
  + yellow
  + transparent
  + white
  + black
* x – postavlja poziciju bloka na x osi; Moguće vrijednosti:
  + 50 – postavlja blok na pixelu 50 gledano slijeva
  + 30.45% - adaptira poziciju na x osi ovisno o širini roditelja
  + left – postavlja blok na lijevu stranu roditelja
  + right – postavlja blok na desnu stranu roditelja
  + center – postavlja blok u sredinu roditelja
* y – postavlja poziciju bloka na y osi; Moguće vrijednosti:
  + 50 – postavlja blok na pixelu 50 gledano slijeva
  + 30.45% - adaptira poziciju na y osi ovisno o visini roditelja
  + top – postavlja blok na vrh roditelja
  + bottom – postavlja blok na dno roditelja
  + center – postavlja visinu bloka u sredinu visine roditelja
* width – postavlja duljinu bloka; Moguće vrijednosti:
  + 120 – postavlja duljinu bloka na 120 pixela
  + 50% - postavlja duljinu bloka na pola širine roditelja
* height – postavlja visinu bloka; Moguće vrijednosti:
  + 120 – postavlja visinu bloka na 120 pixela
  + 50% - postavlja visinu bloka na pola visine roditelja
* font – postavlja font teksta; Moguća vrijednost:
  + url(put\_to\_datoteke)
* font\_size – postavlja veličinu fonta / teksta na zadanu vrijednost; Moguća vrijednost:
  + 30 – postavlja veličinu fonta na 30
* text\_color – postavlja boju teksta; Moguća vrijednost:
  + rgb(crvena, zelena, plava)
  + rgba(crvena, zelena, plava, alpha)
  + red
  + green
  + blue
  + yellow
  + transparent
  + white
  + black
* text\_position – postavlja gdje će tekst biti unutar bloka; Moguće vrijednosti:
  + TopLeft
  + Top
  + TopRight
  + Left
  + Center
  + Right
  + BottomLeft
  + Bottom
  + BottomRight

1. Svrha

Styles podsustav stvoren je s razlogom da pruža stilove u stalnom, definiranom paketu kako bi ga Scenes podsustav mogao iskoristiti da definira atribute blokova. Sa Styles-om nije potrebno definirati stil pojedinačno za svaki blok, već svaki blok može samo uputiti CUI sustavu koji stil želi iskoristiti te, ako postoji, postaviti atribute bloka na atribute stila. Ako stil mijenja iste atribute kao prijašnji stil, atribut će se postaviti na atribut stila koji je novijeg stila.

Situacija se može lako predočiti sljedećom analogijom:   
Zamislimo da smo na servisu za bojanje automobila i zamislimo da su dijelovi našeg automobila, kao što su gume, krov, vozačeva vrata i suvozačeva vrata, njegovi atributi. I zamislimo da su stilovi naše naredbe za bojanje dijelova našeg auta. Te naredbe moramo reći serviseru slijedno te će ih on tako i izvršiti. Zatim zamislimo da smo zadali naredbu da oboja gume u plavo, a krov u crveno. I onda da smo mu zadali da nam vozačeva vrata i suvozačeva vrata oboja u zeleno, ali da gume oboja u crno.

Dogodit će se sljedeće:

Serviser će izvršiti našu prvu naredbu i obojati gume u plavo te krov u crveno. Zatim će izvršiti našu drugu naredbu i obojati vozačeva vrata i suvozačeva vrata u zeleno, ali će prebojiti gume te će gume postati crne.

1. Format

Styles format je jako sličan CSS-ovom formatu datoteke, ali se ipak razlikuje u određenim stavkama. CSS nema pojam o korisničko-definiranim događajima, ali Styles ima. To jest, ima sučelje gdje korisnik može definirati na koje događaje želi postaviti stil. Format ne podržava komentare.

Općeniti format datoteke je:

ime-stila ?[ime-događaja1 ?, ime-događaja2, …?]? {

ime-atributa: vrijednost-atributa;

…

}

…

gdje ?...bilošto…? označava neobavezan dio formata i gdje je:

* ime-stila – alfanumerička vrijednost koja sadrži znakove od [A-Za-z0-9\_] počevši sa slovom ili podvlakom
* ime-događaja – niz znakova koji predstavljaju ime događaja
* ime-atributa – niz znakova koji predstavljaju ime atributa
* vrijednost-atributa – može biti cjelobrojni ili decimalni broj, postotak, funkcija ili riječ
  + primjer:
    - 5
    - 3.14
    - 50%
    - rgba(255, 125, 0, 225)
    - green

Napomena: pri pisanju datoteka potrebno je uokviriti Scene kod u R“(ovdje-ide-kod)“ kako bi se datoteka mogla ugraditi u C++ kod.

1. Parsiranje formata

Iz formata imamo sve informacije koje nam trebaju da stvorimo parser tog formata. Parsiranje je, u osnovi, iteriranje kroz neki skup podataka i provođenje pravila formata te izvlačenje tokena iz tog skupa podataka. U slučaju CUI-a, taj skup podataka je ugrađen niz znakova iz neke zadane datoteke tijekom kompilacije.

U pseudokodu:

parsiraj(ulaz):

dok nije kraj ulaza:

pojedi tok podataka dok ne naiđeš na bilokoji od sljedećih znakova: []:;{}

makni razmake i slično iz pojedenog

ovisno na kojemu je znaku prekinuto jedenje:

ako je na '[':

ako prijašnje stanje nije početak stila, baci grešku

ako je obrađen podatak prazan, baci grešku

ako nije valjano ime, baci grešku

dodaj definiciju

postavi ime zadnje definicije na obrađen podatak

promijeni prijašnje stanje jedenja

ako je na ']':

ako prijašnje stanje nije početak liste događaja, baci grešku

ako je obrađen podatak prazan, baci grešku

ako se dogodila greška tijekom parsiranja liste događaja, baci grešku

promijeni prijašnje stanje jedenja

ako je na '{':

ako prijašnje stanje nije kraj liste događaja i nije početak stila, baci grešku

ako je prijašnje stanje početak stila i ako je obrađen podatak prazan, baci grešku

ako se dogodila greška tijekom parsiranja liste događaja, baci grešku

ako je prijašnje stanje početak stila:

ako nije valjano ime stila, baci grešku

dodaj definiciju

postavi ime zadnje definicije na obrađen podatak

promijeni prijašnje stanje jedenja

ako je na '}':

ako prijašnje stanje nije početak definicije stila, tj. '{' i nije kraj definicije atributa, baci grešku

promijeni prijašnje stanje jedenja

ako je na ';':

ako prijašnje stanje nije početak vrijednosti atributa, baci grešku

ako je obrađen podatak prazan, baci grešku

postavi vrijednost atributa zadnje definicije na obrađen podatak

promijeni prijašnje stanje jedenja

ako je na ':':

ako prijašnje stanje nije početak definicije stila i nije kraj vrijednosti atributa, baci grešku

ako je obrađen podatak prazan, baci grešku

postavi vrijednost atributa zadnje definicije na obrađen podatak

promijeni prijašnje stanje jedenja

vrati tokenizirani rezultat

1. Primjeri datoteka

bgcolor\_red {

background: white;

background: rgba(60, 40, 50, 100);

background: rgb(255, 0, 0); // Vrijednost se postavi na zadnji postavljeni atribut u stilu

}

button {

background: url(./abc.png);

text\_color: red;

font: url(./arial.ttf);

font\_size: 50;

text\_position: center;

x: center;

y: center;

width: 50%;

height: 90;

}

1. Schematic klasa

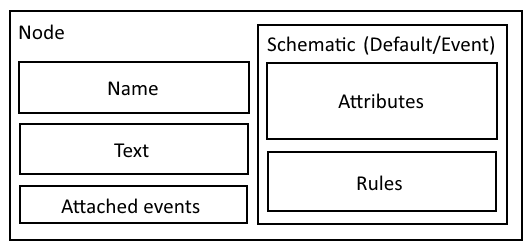
Schematic klasa je klasa koja sadrži sve atribute i sva pravila nekog bloka.

1. Node klasa

Node klasa je element koji SceneGraph klasa sadrži u svojem grafu. Ona je završni oblik Block klase, u kojoj su sheme populirane stavkama iz stilova, u kojoj je ime postavljeno na ime iz bloka i u kojoj je tekst postavljen na tekst iz bloka.

Node klasa ima N+1 broj shema. Jedna je početna shema, a sve ostale su sheme navedenih događaja.

Node klasa također sadrži sučelje za registriranje događaja na tom bloku.



Slika 2. Shema Node klase

1. Scenes podsustav

Scenes je podsustav CUI-a i format koji diktira odnose između blokova u scenskom grafu, određuje koji blok će imati koji stil i koji tekst će blok u sebi sadržati.

1. Svrha

Scenes podsustav stvoren je s namjerom da olakša stvaranje kompleksnih aplikacija sa puno elemenata i da vizualizira odnose između blokova unutar grafa za lakše i brže implementiranje aplikacija.

1. Format

Scene format koristi tabulatore kako bi se označila ovisnost o nekom bloku. Svaki tabulator označava veću razinu. Blokovi na istoj razini su djeca bloku niže razine iz kojega je povećana razina. Blokovi nulte razine su djeca bloku-korijenu (root bloku). Format ne podržava komentare.

Općeniti format datoteke je:

ime-bloka ?"ime-stila1, ..."? ?<tekst-unutar-bloka>?

gdje ?...bilošto…? označava neobavezan dio formata i gdje je:

* ime-bloka – proizvoljno ime bloka
* ime-stila – ime stila kojeg želimo da blok koristi
* tekst-unutar-bloka – tekst koji će biti sadržan i prikazan unutar bloka

Napomena: pri pisanju datoteka potrebno je uokviriti Scene kod u R“(ovdje-ide-kod)“ kako bi se datoteka mogla ugraditi u C++ kod.

1. Parsiranje formata

U pseudokodu:

parsiraj(ulaz):

definiraj stog za indekse blokove

dok nije kraj ulaza:

pojedi tok podataka dok ne naiđeš na novu liniju

makni razmake i slično iz pojedenog

prebroji broj tabulator na početku linije

pronađi prvi apostrof koji indicira početak liste stilova

pronađi prvi simbol lijeve strelice koji indicira početak tekst sekcije

provjeri ako postoji lista stilova tako da:

provjeriš ako pronađen indeks apostrofa nije veći od indeksa lijeve strelice

pronađi zadnji simbol desne strelice koji indicira kraj tekst sekcije

ako je simbol kraja teksta sekcije prije simbola početka teksta sekcije, baci grešku

dodaj u rezultat novi prazni blok

izvuci početni token (ili početak liste stilova ili početak tekst sekcije; ovisno o indeksu)

izvuci ime iz obrađenog podatka pomoću početnog tokena

ako ime bloka nije validno, baci grešku

postavi ime bloka na zadnjem bloku u rezultatu

ako ima liste stilova:

ako nema tekst sekcije:

ako zadnji simbol nije apostrof, baci grešku

ako postoji početak liste stilova:

izvuci drugi token (kraj liste stilova)

ako je početak tekst sekcije prije kraja liste stilova, baci grešku

izvuci listu stilova pomoću početnog i drugog tokena

ako lista stilova nije validna, baci grešku

inače, populiraj listu stilova unutar zadnjeg bloka u rezultatu

ako ima tekst sekcije:

ako zadnji simbol nije simbol desne strelice, baci grešku

kalkuliraj veličinu tekst sekcije

postavi tekst bloka na izvučenu tekst sekciju pomoću indeksa početka-

-tekst sekcije i kalkulirane veličine

ako ima tabulatora na početku:

ako je trenutačan broj tabulatora veći od prijašnjeg za 2 ili više, baci grešku

postavi razinu bloka na trenutačan broj tabulatora

pronađi indeks roditelja pomoću stoga za indekse blokova

ako nema roditelja, baci grešku

dodaj roditelju trenutačan blok kao dijete

dodaj na stog indeks trenutačnog bloka

promijeni prijašnju dubinu na trenutačnu dubinu

inače:

očisti stog indeksa blokova

dodaj na stog indeks trenutačnog bloka

postavi prijašnju dubinu na 0

vrati rezultat

1. Primjeri datoteka

blok1

blok2 "crvena\_boja"

blok3 "rezultat"

blok4 "bijela\_boja, veliki\_gumb" <Klikni me!>

blok1 <Broj jedan>

blok2 "stil2" <Broj dva>

blok3 "stil3" <Broj tri>

text\_box\_container "text\_box\_holder"

text\_box "font, text\_box" <Click on the buttons!>

button1 "font, button, btn\_size, btn1" <Red text>

button2 "font, button, btn\_size, btn2" <Green text>

button3 "font, button, btn\_size, btn3" <Blue text>

1. SceneGraph

SceneGraph je usmjereni graf[[3]](#footnote-3) blokova koji je implementiran kao N-tero stablo. Uz stablo parsiranih blokova, sadrži korijenski-blok. Pruža sučelje za pronalazak roditelja zadanog bloka. Služi kao apstrakcija prozora i njegovih elemenata.

1. CUI-SFML binding

CUI-SFML je binding CUI-a koji iskorištava CUI kako bi se koristio lagano uz SFML biblioteku.

SFML (Simple Fast Multimedia Library) je biblioteka koja apstrahira grafiku u C++ klase.

1. Window klasa

Window klasa je glavna komponenta CUI-SFML-a koja ujedinjuje CUI, grafiku i sustav za događaje. Pruža sučelje za registriranje i odjavljivanje lokalnih i globalnih događaja, vezanje događaja na blokove, odašiljanje normalnih i tajmer događaja, javljanje za obnovu   
RenderCache-a.

Konstruktori uzimaju set stilova i scena kojima generiraju potrebne podatke za prikaz na ekran. Window klasa populira SceneGraph sa blokovima te dodaje to u TrackedList objekt koji drži sve SceneGraph-ove koji su generirani te prati trenutno aktivni graf. U init() funkciji se pruža informacije o prozoru (duljina, visina, ime prozora) i stvara se nova dretva koja obnavlja SceneGraph klasu sa informacijama o korijenskom-bloku, populira podatke unutar RenderCache klase te započinje glavnu petlju prozora.

Funkcija handle\_events() poll-a SFML događaje te ih delegira process\_event() funkciji kako bi se događaj obradio te izvršio korisničko-definirani događaj ukoliko postoji.

Funkcija register\_event() registrira događaj na trenutno aktivnoj sceni. Ukoliko se želi registrirati događaj na nekoj drugoj sceni, potrebno je promijeniti trenutno aktivnu scenu. Prima SFML tip događaja, željeno ime događaja i funkciju koja će se izvršiti tijekom odašiljanja.

Funkcija register\_global\_event() registrira događaj na trenutno aktivnoj sceni, ali se odašilje trenutno ukoliko je SFML događaj istog tipa koji je registriran. Dalje je isto kao i register\_event().

Funkcije unregister\_event() i unregister\_global\_event() odjavljuju određeni događaj te se neće više odašiljati ili gledati.

Funkcija attach\_event\_to\_node() će vezati zadani događaj za blok. Kao parametar prima ime bloka za kojeg se veže događaj i ime događaja koji se veže za blok. Baca iznimku ukoliko blok nije pronađen u aktivnoj sceni.

Funkcija detach\_event\_to\_node() miče vezu događaja za blok. Kao parametar prima ime bloka kojemu se miče veza za događaj i ime događaja čija se veza miče sa bloka. Baca iznimku ukoliko blok nije pronađen u aktivnoj sceni.

Funkcija dispatch\_event() odašilje zadani događaj. Automatski se brine o tome je li događaj globalan ili lokalan. Dvije su verzije ove funkcije, jedna bez dodatnih podataka o događaju, a jedna sa dodatnim podatcima o događaju.

Funkcija process\_event() zaslužna je za populiranje podataka o događaju i o pretragi na kojemu je bloku događaj okinut.

Funkcija timer\_dispatch\_event() odašilje događaj nakon zadanog vremenskog perioda. Događaj se nakon što vremenski period istekne, izvršava na tajmer dretvi.

Funkcija timer\_execute\_next\_in\_line() izvršava sljedeći događaj u redu.

Funkcija timer\_wait\_until\_push() stavlja dretvu u čekanje dok se ne okine novi tajmer događaj.

Funkcija timer\_wait\_for() čeka dok ne dođe vrijeme za izvršavanje događaja sa najkraćim vremenskim periodom.

Funkcija schedule\_to\_update\_cache() zakazuje novu obnovu RenderCache objekta.

Funkcija update\_cache() obnavlja RenderCache objekt.

Funkcija render() prikazuje scenu.

Funkcija resize() mijenja veličinu korijenskog bloka i SFML prozora.

Funkcija close() zatvara prozor.

Destruktor Window klase spaja dretve sa glavnom dretvom, tj. čeka na završetak izvršavanja dretvi.

1. RenderCache

RenderCache je posrednik između CUI-a i SFML-a te izvršitelj pretvorbe iz CUI sustava u SFML sustav. RenderCache je ujedno i grafički spremnik svih blokova trenutačno aktivne scene. Kako bi prikazivao blokove ovisno o njihovoj dubini, RenderCache prvo sortira svoje elemente ovisno o dubini pojedinačnog bloka unutar SceneGraph-a. Također, pri stvaranju prozora, potrebno je predmemorirati određene podatke kao što su teksture i fontovi zbog njihove ovisnosti o OpenGL sustavu (svaki resurs u SFML-u mora biti stvoren u istoj dretvi gdje je i glavna petlja te prozor stvoren).

Funkcija sort() sortira RenderCache ovisno o dubini blokova.

Funkcija cache\_resource() predmemorira određene resurse kako bi aplikacija funkcionirala normalno.

Funkcija update\_ve() obnavlja zadani vizualni element unutar RenderCache-a.

Funkcija update\_root() obnavlja korijenski vizualni element.

Funkcija update\_cache() obnavlja cijeli RenderCache.

Funkcija handle\_background() obrađuje blokovski atribut za pozadinu.

Funkcija handle\_font() obrađuje blokovski atribut za font.

Funkcija handle\_x() obrađuje blokovski atribut za x poziciju.

Funkcija handle\_y() obrađuje blokovski atribut za y poziciju.

Funkcija handle\_width() obrađuje blokovski atribut za širinu.

Funkcija handle\_height() obrađuje blokovski atribut za visinu.

Funkcija handle\_font\_size() obrađuje blokovski atribut za veličinu fonta.

Funkcija handle\_text\_color() obrađuje blokovski atribut za boju teksta.

Funkcija handle\_text\_position() obrađuje blokovsko pravilo za poziciju teksta.

Funkcija handle\_rule\_x() obrađuje blokovsko pravilo za x poziciju.

Funkcija handle\_rule\_y() obrađuje blokovsko pravilo za y poziciju.

Funkcija handle\_rule\_width() obrađuje blokovsko pravilo za širinu.

Funkcija handle\_rule\_height() obrađuje blokovsko pravilo za visinu.

1. EventData

EventData klasa služi za pružanje informacija događaju o tome koji blok je zvao događaj, indeks tog bloka, ime događaja i uniju svih SFML podataka o događaju.

Unija SFML podataka o događaju može sadržati informacije o događaju promjene veličine prozora, o lokaciji kliktanja miša, o pritisku tipke na tipkovnici, itd.

Lista svih spremljenih SFML događaja:

using data\_variant\_t = std::variant<

Empty,

sf::Event::SizeEvent,

sf::Event::KeyEvent,

sf::Event::TextEvent,

sf::Event::MouseMoveEvent,

sf::Event::MouseButtonEvent,

sf::Event::MouseWheelEvent,

sf::Event::MouseWheelScrollEvent,

sf::Event::JoystickConnectEvent,

sf::Event::JoystickMoveEvent,

sf::Event::JoystickButtonEvent,

sf::Event::TouchEvent,

sf::Event::SensorEvent>;

1. TimerEvent

TimerEvent klasa služi za upakiranje funkciju događaja i vremenski period kako bi se moglo lako izvršiti funkciju i kako bi se period mogao uspoređivati sa drugim TimerEvent objektima da se sortiraju u listi.

U Window klasi je tajmer red implementiran kao std::priority\_queue<TimerEvent>.

1. VisualElement

VisualElement klasa je SFML grafički blok. Njega se prikazuje u render() funkciji. On nasljeđuje od sf::View i sf::RectangleShape klasa te sadrži sf::Text kako bi prikazao tekst bloka.

1. WindowOptions

WindowOptions klasa služi kako bismo Window klasi rekli kako kreirati prozor. Određuje širinu i visinu prozora, ime prozora, stil prozora[[4]](#footnote-4) (bez obruba okolo, onemogućiti promjenu veličine prozora, itd.), postavke konteksta te brzine slika (engl. frame rate).

1. Templates

Templates su skup pomoćnih funkcija koje olakšavaju implementiranje određenih funkcionalnosti blokova kao što su klikovi miša ili promjena veličine prozora. Trenutačno postoje 4 pomoćne funkcije u CUI-SFML-u:

* OnClick – izvršava zadanu funkciju ako je blok bio kliknut
* OnHover – izvršava jednu od dviju zadanih funkcija ovisno o tome je li miš na bloku ili ne
* OnResize – zove funkciju resize() iz Window klase te zakazuje novu obnovu RenderCache objekta
* SwitchToDefaultSchematic i SwitchToEventSchematic – postavlja aktivnu shemu ovisno o EventData objektu koji je zadan

1. Intermediaries

Intermediaries su skup klasa koja omogućavaju laku pretvorbu iz CUI tipova podataka u SFML tipove podataka kao što su na primjer cui::Color i sf::Color. To je omogućeno pomoću Proxy klase koja radi kao označena unija svih potrebnih klasa za pretvorbu. Trenutačno je potrebno samo pretvorba između cui::Color i sf::Color klasa, ali se lako proširi funkcionalnost.

1. Primjeri aplikacija
2. Gumb koji mijenja boju pozadine

Stilovi koji se upotrebljavaju:

root {

background: rgb(116, 109, 105); //

}

button {

background: url(button.png);

font: url(bebas\_neue.ttf);

x: center;

y: 70%;

width: 350;

height: 100;

}

Scene koje se upotrebljavaju:

button "button" <Change the background color>

C++ kod koji upogonjuje aplikaciju:

#include <random>

#include <string>

#include <type\_traits>

#include <utility>

#include <cui/compile\_time/scene.hpp>

#include <cui/compile\_time/scenes/parse\_scenes.hpp>

#include <cui/compile\_time/style.hpp>

#include <cui/compile\_time/styles/parse\_styles.hpp>

#include <cui/compile\_time/value\_data.hpp>

#include <cui/visual/node.hpp>

#include <cui/visual/scene\_graph.hpp>

#include <cursors.hpp>

#include <detail/intermediaries/color.hpp>

#include <detail/templates/on\_click.hpp>

#include <detail/templates/on\_resize.hpp>

#include <detail/templates/switch\_schematic.hpp>

#include <visual\_element.hpp>

#include <window.hpp>

#include <window\_options.hpp>

// STATIC\_STRING\_HOLDER je makronaredba koja omogućava lako ugrađivanje  
// teksta iz drugih datoteka

// Omogućuje definiranje posebnog imena za varijablu koju stvori

#define STATIC\_STRING\_HOLDER(name) static constexpr const char name[] =

#define END\_STATIC\_STRING\_HOLDER ;

int main() {

using namespace cui;

// Učitaj styles datoteku

STATIC\_STRING\_HOLDER(style\_\_)

#include "file.styles"

END\_STATIC\_STRING\_HOLDER

// Učitaj scene datoteku

STATIC\_STRING\_HOLDER(scene\_\_)

#include "file.scene"

END\_STATIC\_STRING\_HOLDER

// Aliasi za tipove

using win\_t = Window;

using node\_t = Node;

using event\_data\_t = EventData<node\_t>;

// Pokazivač na naš prozor

std::unique\_ptr<win\_t> window;

// Objekti za nasumičnu generaciju

static auto rand\_device = std::make\_unique<std::random\_device>();

static auto mt\_rand = std::make\_unique<std::mt19937>((\*rand\_device)());

static std::uniform\_int\_distribution rand\_255(0, 255);

// Zaseban okvir kako bi se sve počistilo na kraju

{

// Parsiraj scenu

constexpr auto scenes\_variant = ct::scenes::parse\_scenes<scene\_\_>();

// Ukoliko postoji greška unutar .scene datoteke, ispiši grešku

if constexpr (scenes\_variant.is\_type\_b()) {

println(scenes\_variant.type\_b());

return 0;

}

// Parsiraj stilove

constexpr auto styles\_variant = ct::styles::parse\_styles<style\_\_>();

// Ukoliko postoji greška unutar .styles datoteke, ispiši grešku

if constexpr (styles\_variant.is\_type\_b()) {

println(styles\_variant.type\_b());

return 0;

}

// Inače populiraj Window objekt sa parsiranim podatkom

else

{

// Spremi sve stilove u dinamički niz

std::vector<ct::Style> sty;

sty.reserve(styles\_variant.type\_a().size());

// Iteriraj kroz svaki parsirani stil, kreiraj Style objekt

// i dodaj ga u dinamički niz stilova

for (const auto& el : styles\_variant.type\_a()) {

const auto parsed\_variant = ct::Style::create(el);

// Ukoliko se dogodila greška tijekom kreiranja Style objekta

// ispiši grešku

if (parsed\_variant.is\_type\_b()) {

println(parsed\_variant.type\_b());

return 0;

}

const auto& parsed = parsed\_variant.type\_a();

sty.push\_back(parsed);

}

// Kreiraj Window objekt sa obrađenim podatcima

window = std::make\_unique<win\_t>(std::move(sty),  
 std::move(scenes\_variant.type\_a()));

}

}

// Alias za SFML tipove događaja

using EventType = sf::Event::EventType;

// Registriraj on\_close događaj  
 // OBVEZNO, jer inače se prozor ne može zatvoriti uobičajenim metodama!

window->register\_global\_event(EventType::Closed,   
 "on\_close",  
 [&window](auto event\_data)  
 { window->close(); });

// Registriraj on\_resize događaj

// Opcionalno, ali ukoliko promijenimo veličinu prozora, SceneGraph  
 // se neće obnoviti kako treba

window->register\_global\_event(EventType::Resized,  
 "on\_resize",  
 [&window](auto event\_data)   
 { templates::OnResize((\*window),  
 event\_data);   
 });

// Registriraj on\_click\_btn događaj

// Definira funkcionalnost kada se gumb pritisne

window->register\_event(EventType::MouseButtonPressed,   
 "on\_click\_btn",   
 [&window](event\_data\_t event\_data) {

templates::OnClick((\*window),   
 event\_data,   
 [](Window& window, event\_data\_t& event\_data) {

auto& graph = window.active\_scene().graph();

auto& root = graph.root();

// Generiraj nasumične RGB vrijednosti

const auto r = rand\_255(\*mt\_rand);

const auto g = rand\_255(\*mt\_rand);

const auto b = rand\_255(\*mt\_rand);

// Postavi boju pozadine

root.active\_schematic().get().background() = cui::Color(r, g, b);

// Zakaži obnovu RenderCache objekta

window.schedule\_to\_update\_cache();

});

});

// Veže registrirani događaj on\_click\_btn sa gumb blokom

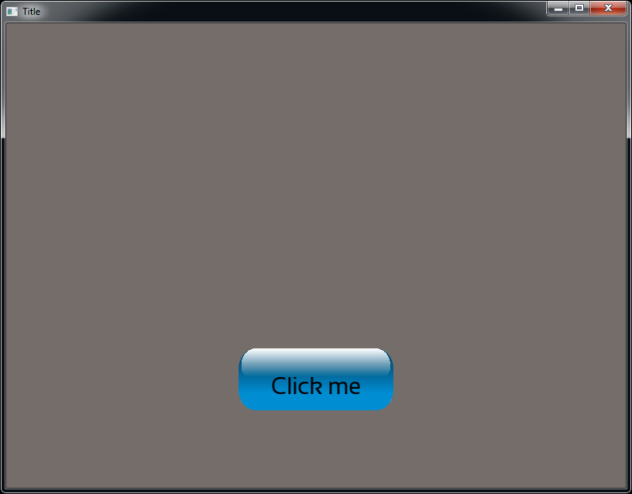
window->attach\_event\_to\_node("button", "on\_click\_btn");

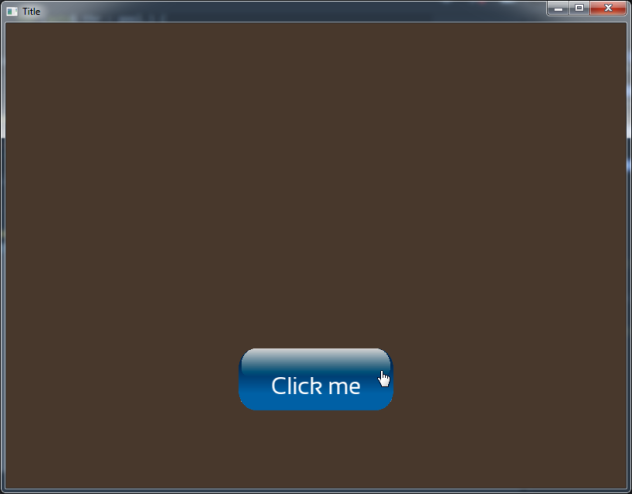
// Inicijaliziraj prozor

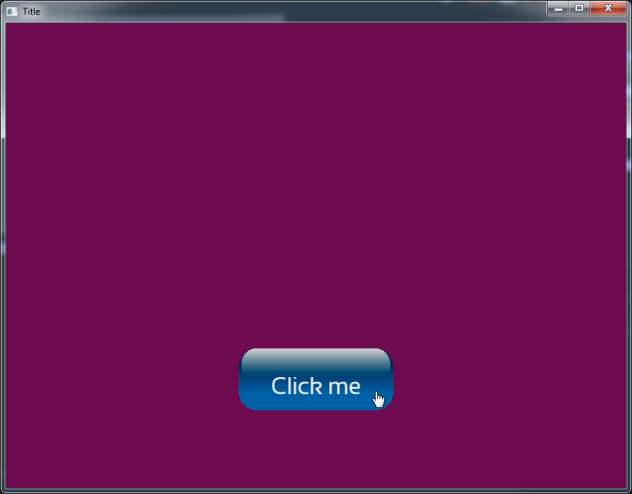
window->init({800, 600,   
 "Title",   
 sf::Style::Default,   
 sf::ContextSettings{},   
 60});

}

 Izgled aplikacije:







1. Više gumbova koji djeluju na isti blok

Stilovi koji se upotrebljavaju:

root {

background: rgb(116, 109, 105);

}

button {

background: url(button.png);

}

font {

font: url(bebas\_neue.ttf);

font\_size: 25;

}

btn\_size {

width: 200;

height: 60;

}

btn1 {

x: left;

y: 70%;

}

btn2 {

x: center;

y: 70%;

}

btn3 {

x: right;

y: 70%;

}

text\_box\_holder {

background: rgb(43, 45, 47);

x: 30%;

y: 50;

width: 40%;

height: 100;

}

text\_box {

background: rgb(192, 192, 192);

x: 10%;

y: 70;

width: 80%;

height: 60;

}

Scene koje se upotrebljavaju:

text\_box\_container "text\_box\_holder"

text\_box "font, text\_box" <Click on the buttons!>

button1 "font, button, btn\_size, btn1" <Red text>

button2 "font, button, btn\_size, btn2" <Green text>

button3 "font, button, btn\_size, btn3" <Blue text>

C++ kod koji upogonjuje aplikaciju:

#include <any>

#include <iostream>

#include <print\_stuff.hpp>

#include <random>

#include <string>

#include <type\_traits>

#include <utility>

#include <cui/compile\_time/scene.hpp>

#include <cui/compile\_time/scenes/parse\_scenes.hpp>

#include <cui/compile\_time/style.hpp>

#include <cui/compile\_time/styles/parse\_styles.hpp>

#include <cui/compile\_time/value\_data.hpp>

#include <cui/utils/print.hpp>

#include <cui/visual/node.hpp>

#include <cui/visual/scene\_graph.hpp>

#include <cursors.hpp>

#include <detail/intermediaries/color.hpp>

#include <detail/templates/on\_click.hpp>

#include <detail/templates/on\_resize.hpp>

#include <detail/templates/switch\_schematic.hpp>

#include <visual\_element.hpp>

#include <window.hpp>

#include <window\_options.hpp>

// STATIC\_STRING\_HOLDER je makronaredba koja omogućava lako ugrađivanje  
// teksta iz drugih datoteka

// Omogućuje definiranje posebnog imena za varijablu koju stvori

#define STATIC\_STRING\_HOLDER(name) static constexpr const char name[] =

#define END\_STATIC\_STRING\_HOLDER ;

int main() {

using namespace cui;

STATIC\_STRING\_HOLDER(style\_\_)

#include "file.styles"

END\_STATIC\_STRING\_HOLDER

STATIC\_STRING\_HOLDER(scene\_\_)

#include "file.scene"

END\_STATIC\_STRING\_HOLDER

// Typedefs for easy use

using win\_t = Window;

using node\_t = Node;

using event\_data\_t = EventData<node\_t>;

std::unique\_ptr<win\_t> window;

// Zaseban okvir kako bi se sve počistilo na kraju

{

// Parsiraj scenu

constexpr auto scenes\_variant = ct::scenes::parse\_scenes<scene\_\_>();

// Ukoliko postoji greška unutar .scene datoteke, ispiši grešku

if constexpr (scenes\_variant.is\_type\_b()) {

println(scenes\_variant.type\_b());

return 0;

}

// Parsiraj stilove

constexpr auto styles\_variant = ct::styles::parse\_styles<style\_\_>();

// Ukoliko postoji greška unutar .styles datoteke, ispiši grešku

if constexpr (styles\_variant.is\_type\_b()) {

println(styles\_variant.type\_b());

return 0;

}

// Inače populiraj Window objekt sa parsiranim podatkom

else

{

// Spremi sve stilove u dinamički niz

std::vector<ct::Style> sty;

sty.reserve(styles\_variant.type\_a().size());

// Iteriraj kroz svaki parsirani stil, kreiraj Style objekt

// i dodaj ga u dinamički niz stilova

for (const auto& el : styles\_variant.type\_a()) {

const auto parsed\_variant = ct::Style::create(el);

// Ukoliko se dogodila greška tijekom kreiranja Style objekta

// ispiši grešku

if (parsed\_variant.is\_type\_b()) {

println(parsed\_variant.type\_b());

return 0;

}

const auto& parsed = parsed\_variant.type\_a();

sty.push\_back(parsed);

}

// Kreiraj Window objekt sa obrađenim podatcima

window = std::make\_unique<win\_t>(std::move(sty),  
 std::move(scenes\_variant.type\_a()));

}

}

// Alias za SFML tipove događaja

using EventType = sf::Event::EventType;

// Registriraj on\_close događaj  
 // OBVEZNO, jer inače se prozor ne može zatvoriti uobičajenim metodama!

window->register\_global\_event(EventType::Closed,   
 "on\_close",  
 [&window](auto event\_data)  
 { window->close(); });

// Registriraj on\_resize događaj

// Opcionalno, ali ukoliko promijenimo veličinu prozora, SceneGraph  
 // se neće obnoviti kako treba

window->register\_global\_event(EventType::Resized,  
 "on\_resize",  
 [&window](auto event\_data)   
 { templates::OnResize((\*window),  
 event\_data);   
 });

// Get the text\_box node

auto& graph = window->active\_scene().graph();

auto text\_box = std::find\_if(graph.begin(),   
 graph.end(),  
 [](const auto& node) {   
 return node.data().name() == "text\_box";   
 });

// Registriraj on\_click\_btn događaj

// Definira funkcionalnost kada se gumb pritisne

window->register\_event(EventType::MouseButtonPressed,   
 "on\_click\_btn",   
 [&window, &text\_box](event\_data\_t event\_data) {

templates::OnClick((\*window),   
 event\_data,   
 [&text\_box](Window& window,   
 event\_data\_t& event\_data) {

// Moguće konstante boja

constexpr cui::Color colors[] =   
 {  
 cui::Color(255, 0, 0),   
 cui::Color(0, 255, 0),   
 cui::Color(0, 0, 255)  
 };

// Moguće konstante tekstova

constexpr cui::ct::StringView texts[] =

{  
 "Red text",   
 "Green text",   
 "Blue text"  
 };

auto& graph = window.active\_scene().graph();

auto& scheme = text\_box->data().active\_schematic().get();

// Dobij identifikator gumba koji je kliknut tako da se  
 // može pristupiti točnom elementu u konstantnom nizu

const auto id = event\_data.caller()->name().back() - '0' - 1;

// Postavi tekst i boju teksta na zadanu vrijednost

scheme.text\_color() = colors[id];

text\_box->data().text() = texts[id].data();

// Zakaži obnovu RenderCache objekta

window.schedule\_to\_update\_cache();

});

});

// Veže registrirani događaj on\_click\_btn sa gumbovima

window->attach\_event\_to\_node("button1", "on\_click\_btn");

window->attach\_event\_to\_node("button2", "on\_click\_btn");

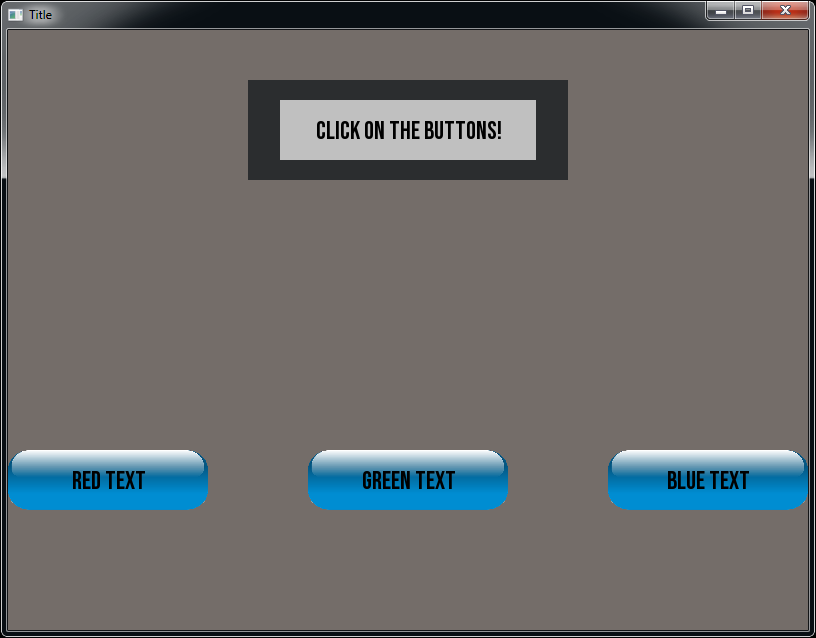
window->attach\_event\_to\_node("button3", "on\_click\_btn");

// Inicijaliziraj prozor

window->init({800, 600,   
 "Title",   
 sf::Style::Default,   
 sf::ContextSettings{},   
 60});

}

Izgled aplikacije:









1. Tipkovnica

Stilovi koji se upotrebljavaju:

root {

background: url(matte\_blue.jpg);

}

font {

font: url(bebas\_neue.ttf);

}

text\_box\_style {

width: 400;

height: 100;

x: 200;

y: 50;

background: transparent;

text\_color: white;

}

container {

y: 200;

height: 400;

}

button {

background: url(button.png);

width: 50;

height: 50;

}

clear\_btn {

background: url(button.png);

x: 250;

y: 70%;

width: 290;

height: 50;

}

b1 { x: 250; y: 0%; }

b2 { x: 310; y: 0%; }

b3 { x: 370; y: 0%; }

b4 { x: 430; y: 0%; }

b5 { x: 490; y: 0%; }

b6 { x: 250; y: 50%; }

b7 { x: 310; y: 50%; }

b8 { x: 370; y: 50%; }

b9 { x: 430; y: 50%; }

b0 { x: 490; y: 50%; }

Scene koje se upotrebljavaju:

text\_box "text\_box\_style, font" <Enter from keypad>

keypad\_container "container"

one "button, font, b1" <1>

two "button, font, b2" <2>

three "button, font, b3" <3>

four "button, font, b4" <4>

five "button, font, b5" <5>

six "button, font, b6" <6>

seven "button, font, b7" <7>

eight "button, font, b8" <8>

nine "button, font, b9" <9>

zero "button, font, b0" <0>

clear "font, clear\_btn" <Clear>

C++ kod koji upogonjuje aplikaciju:

#include <any>

#include <iostream>

#include <print\_stuff.hpp>

#include <random>

#include <string>

#include <type\_traits>

#include <utility>

#include <cui/compile\_time/scene.hpp>

#include <cui/compile\_time/scenes/parse\_scenes.hpp>

#include <cui/compile\_time/style.hpp>

#include <cui/compile\_time/styles/parse\_styles.hpp>

#include <cui/compile\_time/value\_data.hpp>

#include <cui/utils/print.hpp>

#include <cui/visual/node.hpp>

#include <cui/visual/scene\_graph.hpp>

#include <cursors.hpp>

#include <detail/intermediaries/color.hpp>

#include <detail/templates/on\_click.hpp>

#include <detail/templates/on\_resize.hpp>

#include <detail/templates/switch\_schematic.hpp>

#include <visual\_element.hpp>

#include <window.hpp>

#include <window\_options.hpp>

// STATIC\_STRING\_HOLDER je makronaredba koja omogućava lako ugrađivanje  
// teksta iz drugih datoteka

// Omogućuje definiranje posebnog imena za varijablu koju stvori

#define STATIC\_STRING\_HOLDER(name) static constexpr const char name[] =

#define END\_STATIC\_STRING\_HOLDER ;

int main() {

using namespace cui;

STATIC\_STRING\_HOLDER(style\_\_)

#include "file.styles"

END\_STATIC\_STRING\_HOLDER

STATIC\_STRING\_HOLDER(scene\_\_)

#include "file.scene"

END\_STATIC\_STRING\_HOLDER

// Typedefs for easy use

using win\_t = Window;

using node\_t = Node;

using event\_data\_t = EventData<node\_t>;

std::unique\_ptr<win\_t> window;

// Zaseban okvir kako bi se sve počistilo na kraju

{

// Parsiraj scenu

constexpr auto scenes\_variant = ct::scenes::parse\_scenes<scene\_\_>();

// Ukoliko postoji greška unutar .scene datoteke, ispiši grešku

if constexpr (scenes\_variant.is\_type\_b()) {

println(scenes\_variant.type\_b());

return 0;

}

// Parsiraj stilove

constexpr auto styles\_variant = ct::styles::parse\_styles<style\_\_>();

// Ukoliko postoji greška unutar .styles datoteke, ispiši grešku

if constexpr (styles\_variant.is\_type\_b()) {

println(styles\_variant.type\_b());

return 0;

}

// Inače populiraj Window objekt sa parsiranim podatkom

else

{

// Spremi sve stilove u dinamički niz

std::vector<ct::Style> sty;

sty.reserve(styles\_variant.type\_a().size());

// Iteriraj kroz svaki parsirani stil, kreiraj Style objekt

// i dodaj ga u dinamički niz stilova

for (const auto& el : styles\_variant.type\_a()) {

const auto parsed\_variant = ct::Style::create(el);

// Ukoliko se dogodila greška tijekom kreiranja Style objekta

// ispiši grešku

if (parsed\_variant.is\_type\_b()) {

println(parsed\_variant.type\_b());

return 0;

}

const auto& parsed = parsed\_variant.type\_a();

sty.push\_back(parsed);

}

// Kreiraj Window objekt sa obrađenim podatcima

window = std::make\_unique<win\_t>(std::move(sty),  
 std::move(scenes\_variant.type\_a()));

}

}

constexpr auto text\_box\_index = 0;

// Za ponovno ispitivanje „Enter from Keypad“ poruke  
 static bool is\_first\_time = true;

// Alias za SFML tipove događaja

using EventType = sf::Event::EventType;

// Registriraj on\_close događaj  
 // OBVEZNO, jer inače se prozor ne može zatvoriti uobičajenim metodama!

window->register\_global\_event(EventType::Closed,   
 "on\_close",  
 [&window](auto event\_data)  
 { window->close(); });

// Registriraj on\_resize događaj

// Opcionalno, ali ukoliko promijenimo veličinu prozora, SceneGraph  
 // se neće obnoviti kako treba

window->register\_global\_event(EventType::Resized,  
 "on\_resize",  
 [&window](auto event\_data)   
 { templates::OnResize((\*window),  
 event\_data);   
 });

// Registriraj on\_click\_keypad\_btn događaj

// Kada se jedan od tipki na tipkovnici pritisne, dodaje se tekst

// u text\_box-u

window->register\_event(EventType::MouseButtonPressed,   
 "on\_click\_keypad\_btn",   
 [&window, &text\_box](event\_data\_t event\_data) {

templates::OnClick((\*window),   
 event\_data,   
 [&text\_box](Window& window,   
 event\_data\_t& event\_data) {

auto& graph = window.active\_scene().graph();

auto& node = graph[text\_box\_index].data();

// Ako je text\_box sadrži početni tekst, isprazni tekst

if (is\_first\_time) node.text() = "";

// Više nije početni tekst

is\_first\_time = false;

// Pridodaj tekst bloka koji je pritisnut

node.text().append(event\_data.caller()->text());

// Zakaži obnovu RenderCache objekta  
 window.schedule\_to\_update\_cache();

});

});

// Registriraj on\_click\_clear\_btn događaj

// Briše sadržaj unutar text\_box-a te resetira na početno stanje

window->register\_event(EventType::MouseButtonPressed,   
 "on\_click\_clear\_btn",   
 [&window](event\_data\_t event\_data) {

templates::OnClick((\*window),   
 event\_data,   
 [](Window& window, event\_data\_t& event\_data) {

auto& graph = window.active\_scene().graph();

auto& node = graph[text\_box\_index].data();

// Vrati početni tekst

node.text() = "Enter from keypad";

is\_first\_time = true;

// Zakaži obnovu RenderCache objekta

window.schedule\_to\_update\_cache();

});

});

// Veže registrirani događaj on\_click\_btn sa gumbovima

window->attach\_event\_to\_node("one", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("two", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("three", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("four", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("five", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("six", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("seven", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("eight", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("nine", "on\_click\_keypad\_btn");

window->attach\_event\_to\_node("zero", "on\_click\_keypad\_btn");

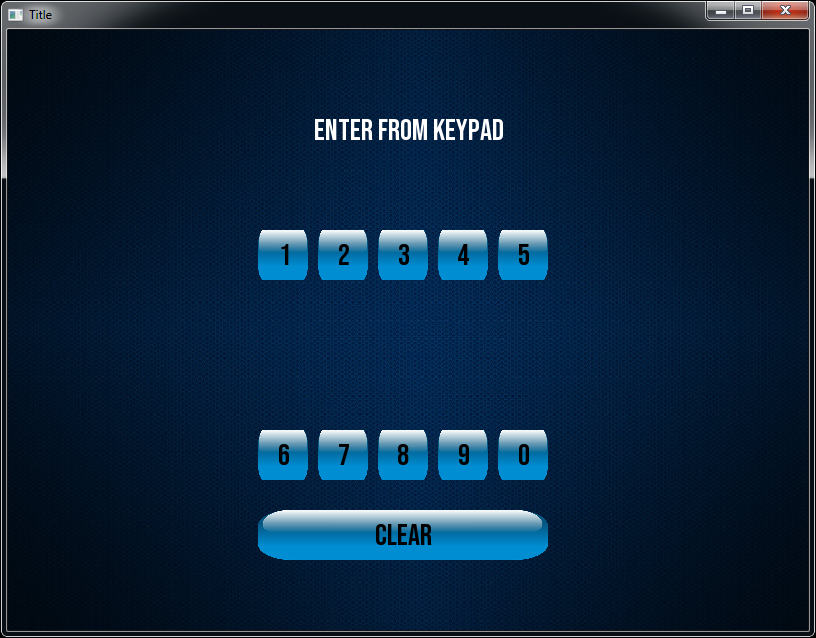
window->attach\_event\_to\_node("clear", "on\_click\_clear\_btn");

// Inicijaliziraj prozor

window->init({800, 600,   
 "Title",   
 sf::Style::Default,   
 sf::ContextSettings{},   
 60});

}

 Izgled aplikacije:





1. Zaključak

Kroz projekt CreateUI naučio sam puno stvari vezano uz to kako GUI sustavi rade, kako održavati veliki projekt te unaprijedio sam svoje znanje o C++-u. Ovaj projekt me natjerao da razmišljam kritički o dizajnu sučelja, formata datoteka i API[[5]](#footnote-5) dizajna. Naučio me koliko je važno prerađivati kod kako bi se usavršavao kod. Tijekom izrade CUI-a, prerađivao sam cijeli projekt 7 puta. I još uvijek smatram da bi moglo biti bolje.

Nekoliko ideja su mi pale na pamet nakon izrade projekta kao što je sustav poslova tako da se aplikacija može izvršavati na više dretva, ili kao što je bogatija funkcionalnost unutar Styles sustava. Primjerice omogućavanje pristupa ostalim atributima i atributima roditelja u nekom drugom atributu. Možda čak i jezik za programiranje događaja kako bi se minimizirao pisanje donekle kompleksnog i velikog dijela C++ koda.

Odabrao sam ovaj projekt jer sam smatrao da ga mogu napraviti i htio sam i nakon završnog raditi na njemu. Vjerujem kako u današnje vrijeme ne postoje adekvatna rješenja za GUI sustave po mojem mišljenju pa sam htio donijeti na svijet nešto novo, nešto bolje. Naravno, da bi se to ostvarilo trebao bih još puno raditi na projektu.

Zahvalio bih se svojem razredniku Zoranu Dumančiću na poštivanju i na načinu njegova predavanja i odrade nastave. Zbog njega sam uvijek ostao donekle aktivan u svojim radovima. Također bih se zahvalio mentoru Goranu Gomazu što me potaknuo na kreativnost i razmišljanje izvan kutije.

1. engl. parser – inform. program koji obavlja sintaktičku analizu nekog jezika; sintaktički analizator [↑](#footnote-ref-1)
2. engl. binding – u kontekstu softverskih biblioteka označuje povezivanje različitih platformi ili jezika na biblioteku [↑](#footnote-ref-2)
3. Usmjereni graf je u matematici struktura povezanih objekata. [↑](#footnote-ref-3)
4. sf::Style enumeracija sadrži bit masku svakog stila

   Više o tome može se naći u dokumentaciji za SFML [↑](#footnote-ref-4)
5. Application Program Interface je sučelje nekog sustava koje korisnik može iskoristiti da postigne neki cilj. [↑](#footnote-ref-5)