Vizualizacija robotskog lica

Šćurec, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Marko Šćurec

Zagreb, 2014. godina.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Prof. dr. sc. Bojan Jerbić, dipl. ing.
Bojan Šekoranja, mag.ing.mech

Student:

Marko Šćurec

Zagreb, 2014. godina.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečeno znanje tijekom studija i navedenu literaturu.

Zahvaljujem se profesoru Bojanu Jerbiću i asistentu Bojanu Šekoranji za mentorstvo nad radom, gđi. Izidori Herold za informacije oko detalja rada i drugim djelima, anonimnoj osobi 'Sinious' s foruma podrške za Adobe programe, te svim ostalima koji su mi pomogli steći znanje, iskustvo, podršku i motivaciju za izradu rada, uključujući vlastitu obitelj i poznanike.

Marko Šćurec

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Marko Šćurec** Mat. br.: 0035178085

Naslov rada na hrvatskom jeziku: **Vizualizacija robotskog lica**

Naslov rada na engleskom jeziku: **Visualization of robotic face**

Opis zadatka:

U sklopu zadataka potrebno je izraditi program za vizualizaciju izraza pojednostavljenog lica, sličnog ljudskom, putem kojeg će se dodatno izražavati ponašanje robota. Potrebno je osmisliti i oblikovati nekoliko animiranih stanja robotskog lica koje se mogu povezati s odabranim signalima stanja robota i pridruženog vizijskog sustava. Predefinirana stanja lica se prikazuju na zaslonu s obzirom na aktivnosti robota i njegovih zahtjeva prema korisniku. Povezivanje programa za vizualizaciju lica i robota treba ostvariti korištenjem standardnih mrežnih protokola. Program mora biti primjenljiv na različitim računalnim platformama - računalo, tablet ili mobitel.

Zadatak implementirati na robotima u Laboratoriju za projektiranje izradbenih i montažnih sustava.

Zadatak zadan:
25. studenog 2014.

Zadatak zadao:
Prof. dr. sc. Bojan Jerbić

Rok predaje rada:
1. rok: 26. veljače 2015.
2. rok: 17. rujna 2015.

Predviđeni datumi obrane:
1. rok: 2., 3., i 4 . ožujka 2015.
2. rok: 21., 22., i 23. rujna 2015.

Predsjednik Povjerenstva:
Prof. dr. sc. Zoran Kunica

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Kao zadatak je potrebno izraditi računalni program koji vizualno prikazuje 'lice' robota, korištenjem vektorskih likova i animacija. Prikazani likovi i animacije pritom se temelje na ulaznim signalima koje program dobiva od robota s kojim je mrežno povezan.

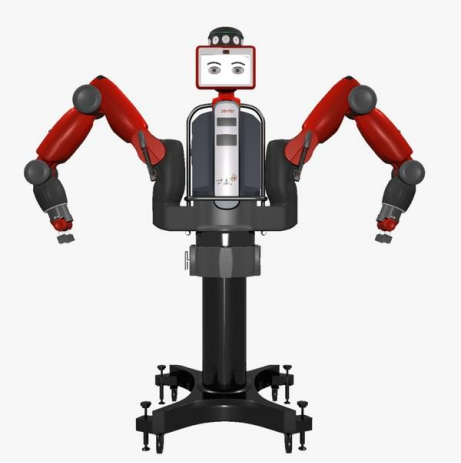
Za mrežno povezivanje programa i robota koristiti će se standardni mrežni protokoli, pri čemu je programu pridodana uloga servera, tj. poslužitelja, koji se spaja s robotom, u ulozi korisnika.

Ključne riječi: Vizualizacija, robotika, simulacija;

1. UVOD

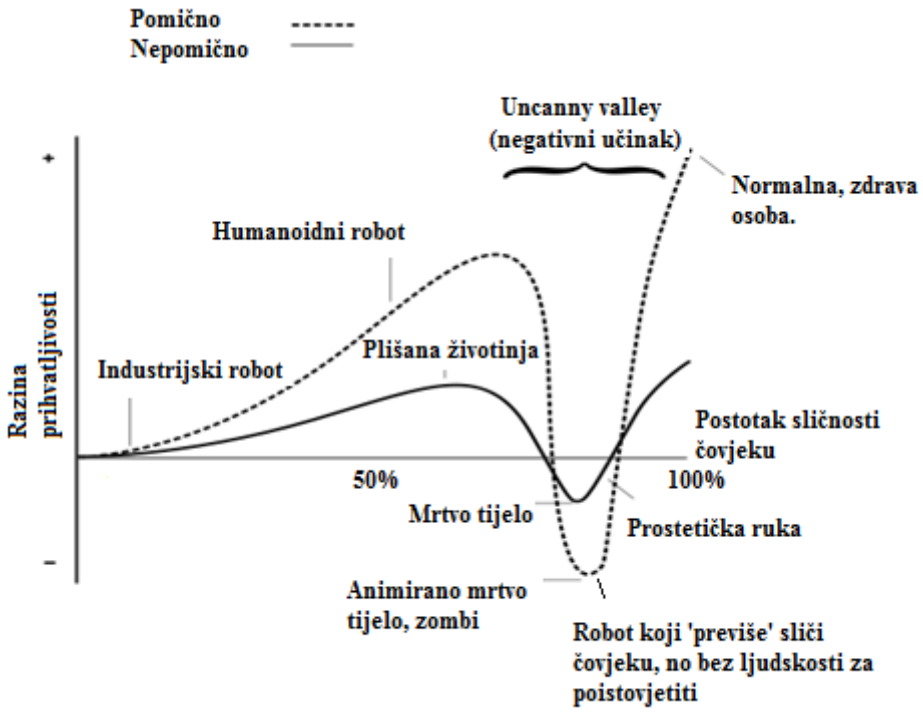
Poznato je da ljudi bolje ostvaruju interakcije pri radu s robotima, ukoliko su na robotu prisutne značajke koje ljudi bolje razumiju. Jednostavnije je razumjeti robote kada imaju značajke slične ljudima [1]. Iako su u osnovi samo alat, roboti se mogu povezati s ljudskim značajkama, pošto im je svrha obavljati ljudske zadaće ili radnje, te su često oblikovani u humanoidnom obličju (nalik čovjeku). Ljudske značajke također mogu pomoći pri upravljanju samih robota: značajke koje su na prvi pogled svima prepoznatljive, poput emocija i sličnih vizualnih elementa, uvelike olakšavaju rad sa strojevima, te ne zahtijevaju ikakvo predznanje. To je iznimna prednost kada se uzme u obzir kompliciranost programskih jezika te logičkog jezika računala - binarnog koda.

Inženjerska struka je prepuna industrijskih robota. Problem kod istih je što su za svrhu funkcionalnosti, maksimalne učinkovitosti pri radu, te ekonomskih prednosti, oblikovani kao kruti alat, s malo ili bez ikakvih ljudskih značajki. Iako s razvojem robotike dolazi i era novih industrijskih robota koji imaju ljudske značajke, kao primjerice model industrijskog robota s monitorom za prikaz lica, zvan Baxter [Slika 1][2], trenutno nije jednostavno niti jeftino utjecati na već postojeće industrijske robote.



Slika 1. Robot 'Baxter' - industrijski robot oblikovan s ljudskim značajkama.

Pritom je važno naglasiti da takve iste značajke mogu imati negativne posljedice za korisnike. Tu se može spomenuti teorija popularno nazvana *'uncanny valley*' [3] [Slika 1]. Potrebno je uspostaviti ravnotežu između prepoznatljivih ljudskih značajki i 'razine prihvatljivosti' robota, inače bi robot mogao izazivati negativne reakcije od korisnika.



Slika 2. Graf *'uncanny valley'* prikazuje moguću negativnu percepciju / reakciju na robote s ljudskim značajkama. Industrijski roboti ne izazivaju reakcije, dok elementi na dnu grafa imaju negativan učinak. [3]

Cilj ovog zadatka jest da se jednostavno, te fleksibilno, uvedu ljudske značajke na bilo kojem odabranom robotu.

Najjednostavnije rješenje jest da se izradi program koji korisniku robota može prikazati jednostavno, animirano lice - popularni 'smiješak' (*smiley*). Lik smiješka podrazumijeva razumljive izraze lica, bez negativnih reakcija ili učinka na korisnika, te korisnik ne treba predznanje o programiranju ili o značajkama robota da bi razumio i koristio informacije koje mu se prikazuju.

2. IZRADA VISUALNIH ELEMENATA - LICE, OČI, DODATNI PROZORI, ANIMACIJE

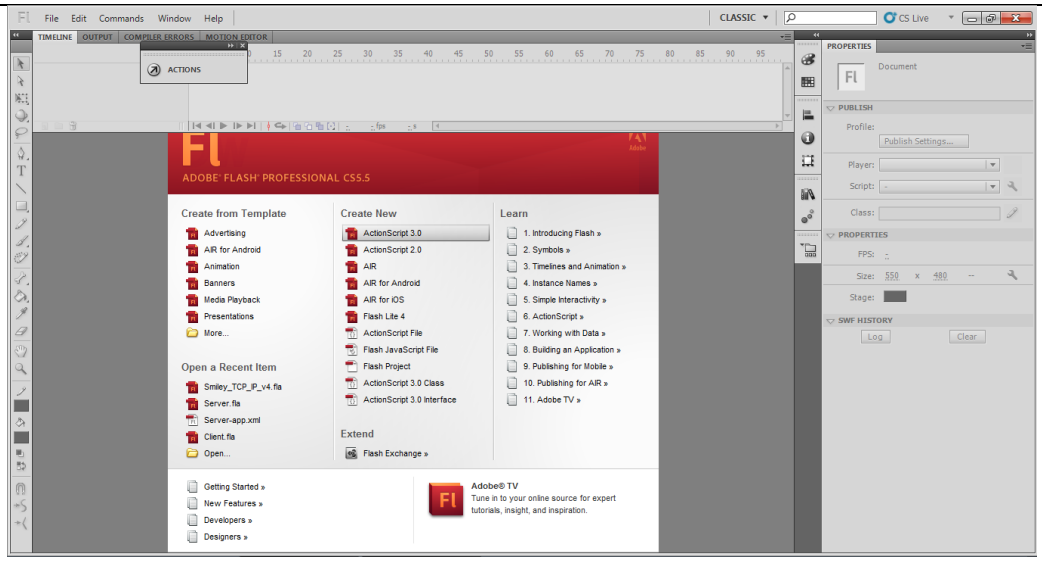
2.1. Odabir razvojnog okruženja

Za izradu programa nužno je odabrati pravilno razvojno okruženje, što u biti određuje osnovnu arhitekturu, mogućnosti, te programske zahtjeve. Pritom je nužno da program ima mogućnost programiranja osnovne geometrije, likova i animacija, čime bi se moglo prikazati 'lice', te da ima programsku mogućnost za mrežno povezivanje. Također je nužno da program bude fleksibilan što se tiče različitih operativnih sustava, internetskih pretraživača, te da je nezavisan od drugih programa.

Da bi se ostvarili ovi uvjeti, odlučeno je da se koristi Flash razvojno okruženje, koje ima mogućnost izrade vektorskih animacija i grafika (za izradu 'lica'), te mogućnosti korištenja mrežnih protokola za uspostavu mrežne veze.

2.2. Odabir alata za programiranje i programskog jezika

Nakon odabira razvojnog okruženja, treba odabrati i alate za izradu programa. U tu svrhu se odabire tzv. Flash Professional CS5.5 [Slika 2], jedan od skupine alata za izradu različitih vrsta Adobe programa [4], u ovom slučaju za Flash radno okruženje. Treba odabrati i programski jezik, koji određuje mogućnosti, pravila, i način kodiranja programa, te razne tehnološke mogućnosti. Odabire se Actionscript 3 [5] kao programski jezik.



Slika 3. Flash Professional CS5.5 alati.

2.3. Objašnjenje sučelja alata

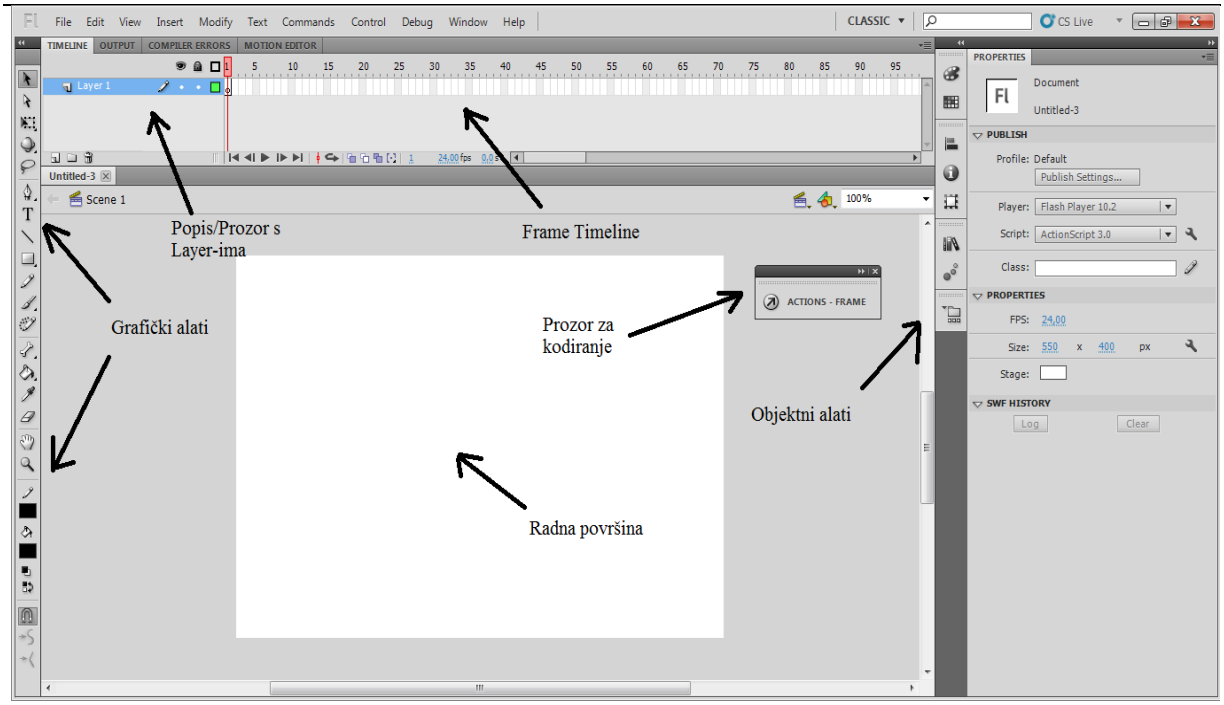
Sučelje [Slika 3] je, u osnovi, alat za izradu animacija. Na lijevoj traci sučelja su razni osnovni alati za crtanje, izradu, i manipulaciju grafičkih elemenata, dok su na desnoj strani alati za objekte i likove.

U samoj sredini programa je radna površina, koja prikazuje vidljivu površinu za trenutno odabrani sloj.

Gornji lijevi prozor prikazuje popis *layer*-a, individualnih 'slojeva' papira, tj. slojeva radnih površina. Oni služe da olakšaju izradu grafičkih elemenata, određuju 'raspored' objekata (tj. koji likovi su ispred ili iza drugih likova), izoliraju dijelove animacija od drugih animacija, te su u stanju sadržavati programski kod kojim se upravlja dijelom ili cijelim programom.

Desno od popisa slojeva je raspored vremenskih okvira (*frame timeline*), sučelje koje prikazuje sve vremenske okvire (*time frame, frame*) neke animacije, za pojedini sloj. Ono omogućava manipulaciju pojedinih trenutaka animacije, te upravljanje prikaza/ponašanja same animacije. Ujedno dozvoljava istovremeno animiranje na više različita načina/slojeva.

Prozor za kodiranje omogućuje upis programskog koda za pojedini okvir, ili za skupinu okvira.

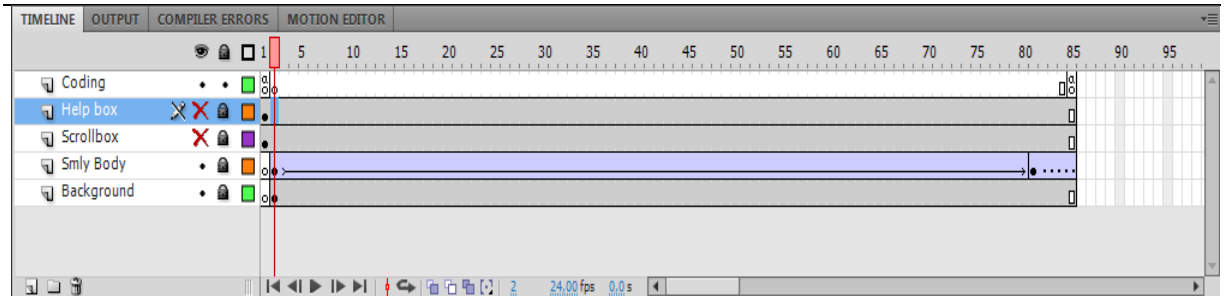


Slika 4. Prikaz Actionscript 3 sučelja.

2.4. Popis i podjela slojeva

Za urednost, jednostavnost, i preglednost rada, različiti elementi programa su razvrstani u vlastite, osnovne slojeve [Slika 4]. Oni su, redom, nazvani:

- *'Coding'*, u kojem je sažeta većina programskog koda za program. Sloj upravlja ponašanjem drugih slojeva kroz nekoliko ključnih vremenskih okvira (*keyframe*).
- *'Help box'*, koji sadrži prozor za pomoć/upute korištenja.
- *'Scrollbox'*, u kojemu je prozor za obavijesti programa.
- *'Smly Body'*, u kojemu su objekti lica. Sloj se grana u daljnje pod-slojeve za pojedine elemente oči i usta lika (koji nadalje imaju svoje vlastito 'grananje' objekata).
- *'Background'*, koji sadrži jednostavnu sivu podlogu, smještenu iza ostalih elemenata.



Slika 5. Glavni slojevi, te njihovi ključni vremenski okviri (crne točke).

3. IZRADA VISUALNIH ELEMENATA - LICE, OČI, DODATNI PROZORI, ANIMACIJE

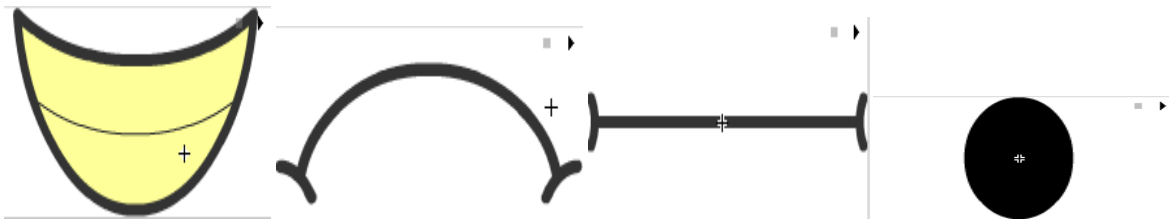
Dok je izrada podloge iznimno jednostavna, izrada ostalih elemenata, naročito onih animiranih, je kompliciranija. U ovom poglavlju će se detaljno proučiti izrada animacijskih elemenata, što je preduvjet izrade programskog koda za iste.

3.1. Oblikovanje 'lica' - oči i usta

Za svrhu prikazivanja stanja robota, određeno je da će se lik 'lica' sastojati od najosnovnijih elementa lica, kojima se mogu predočiti 'emocije'. Time se lice sastoji od dva osnovna objekta, usta, te para oči. Da bi se ti objekti mogli programirati, trebaju se najprije izraditi koristeći geometrijske i grafičke alate, potom se za njih treba odrediti i izraditi 'kostur' animacije - izmjene lika u vremenu, koristeći raspored okvira.

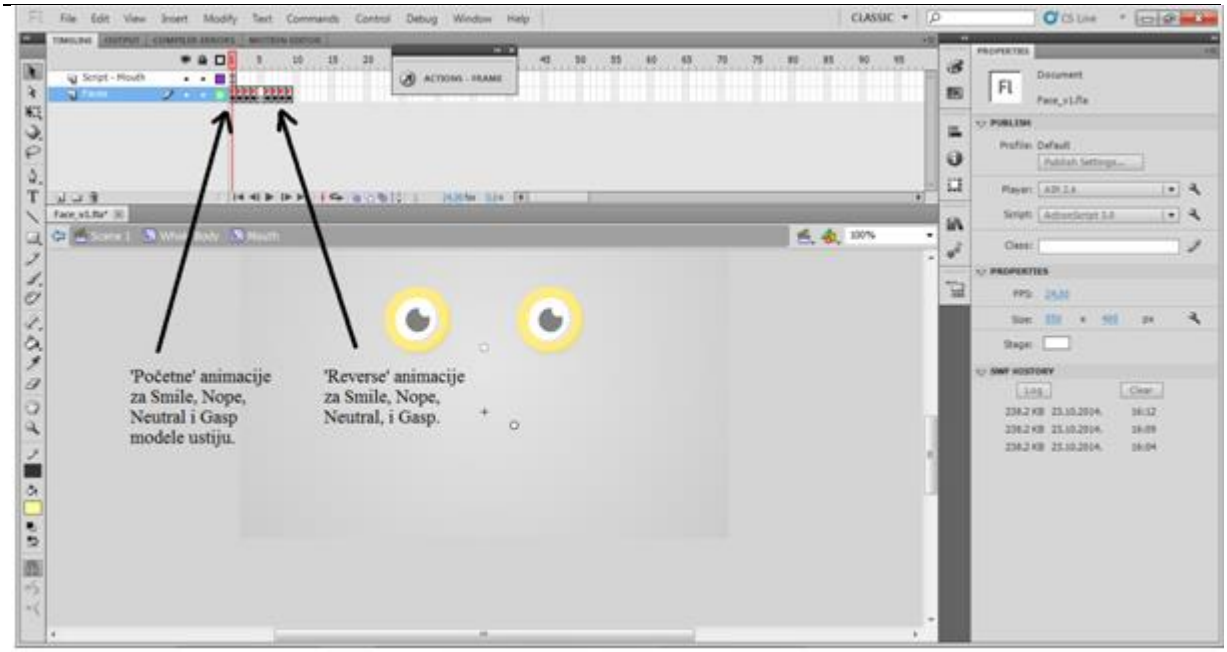
3.2. Izrada modela i animacija usta

Usta su, uz oči, osnovni element za vizualno predočiti izraze lica. Usta se animiraju, pri čemu izmjenjuju četiri moguća modela, svaki od kojih prikazuje emociju, svaki sa svojim 'imenom' - smiješak (*Smile*), nezadovoljstvo (*Nope*), neutralnost (*Neutral*), te čuđenje **(***Gasp***)** [Slika 6].



Slika 6. Grafički modeli/objekti usta - smiješak (*Smile*), nezadovoljstvo (*Nope*), neutralnost (*Neutral*), te čuđenje (*Gasp*).

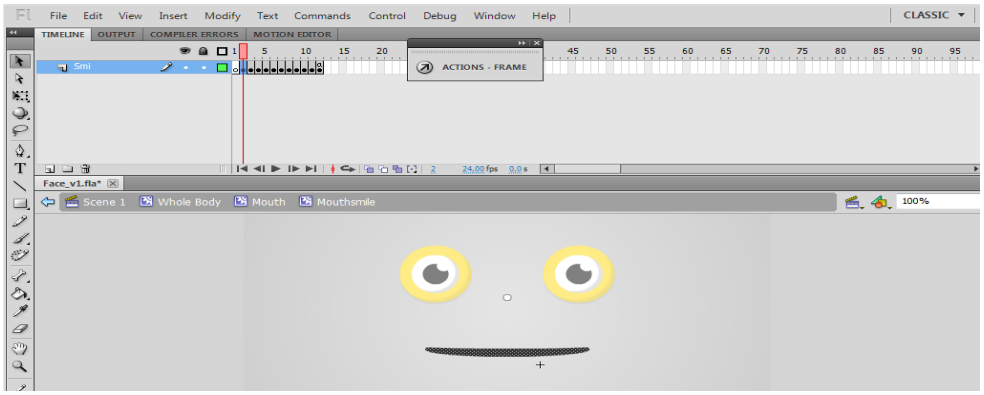
Za svaki od ova četiri modela je potreban raspored okvira za animaciju, pošto je poželjno da lik 'realistično' prelazi tranziciju između različitih prikaza usta. Stoga, unutar pod-slojeva usta, prisutna su osam različita rasporeda animiranja [Slika 7], od kojih su četiri za animaciju 'otvaranja usta' po modelu, dok su ostala četiri za 'zatvaranje' usta, tj. obratnog (*reverse*) postupka.



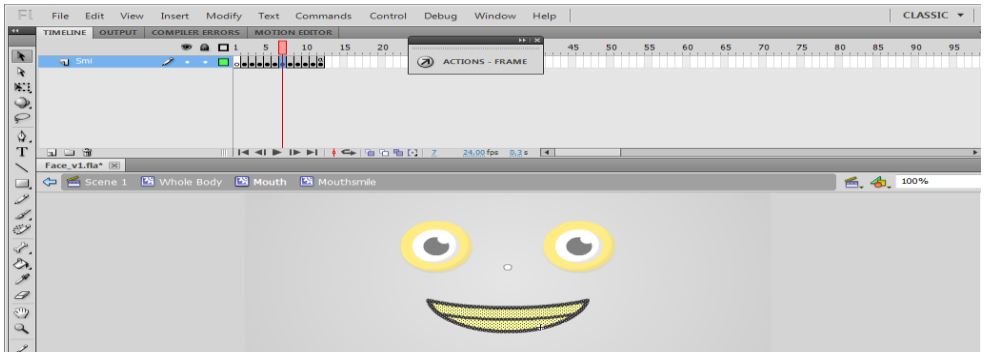
Slika 7. Raspored okvira koji sadrži animacije za sva četiri modela usta, u oba 'smjera'.

Ključnom okviru je najprije zadano ime. To ime će se kasnije koristiti u programskom kodu za pozivanje tog specifičnog dijela animacije, u tom trenutku. Tom ključnom okviru je potom priložen jedan od modela usta, te se stvara pod-sloj unutar kojega se izrađuje animacija za taj model.

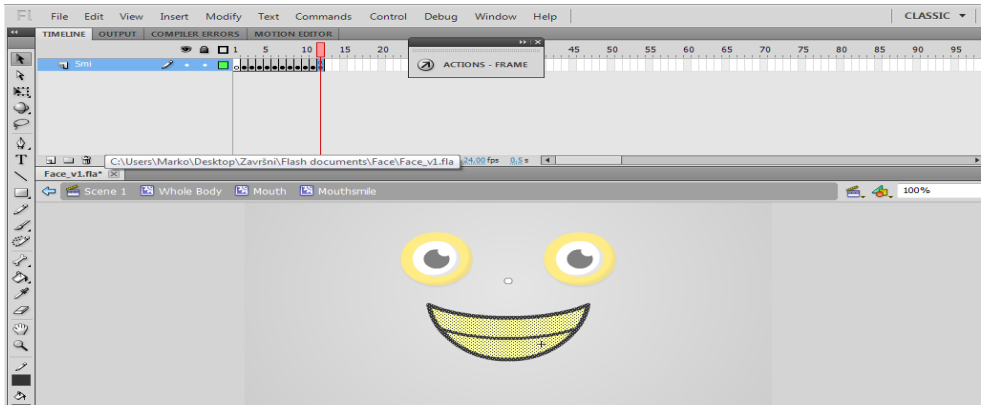
Sama animacija je jednostavna - sastoji se od 12 okvira sveukupno, od kojih je prvi 'prazan', tj. ne prikazuje ništa. U sljedećem (drugom) okviru [Slika 8], usta su prikazana, no umanjena i izobličena. Kroz daljnje okvire [Slika 9] usta se povećavaju, te napokon u zadnjem okviru [Slika 10] poprimaju konačni izgled. Kada se svih 12 okvira posloži u red, stvara se iluzija da usta 'nastaju' iz praznine, tj. da lik 'otvara usta' u vremenu manjem od sekunde.



Slika 8. Prvi okvir s deformiranim likom usta.

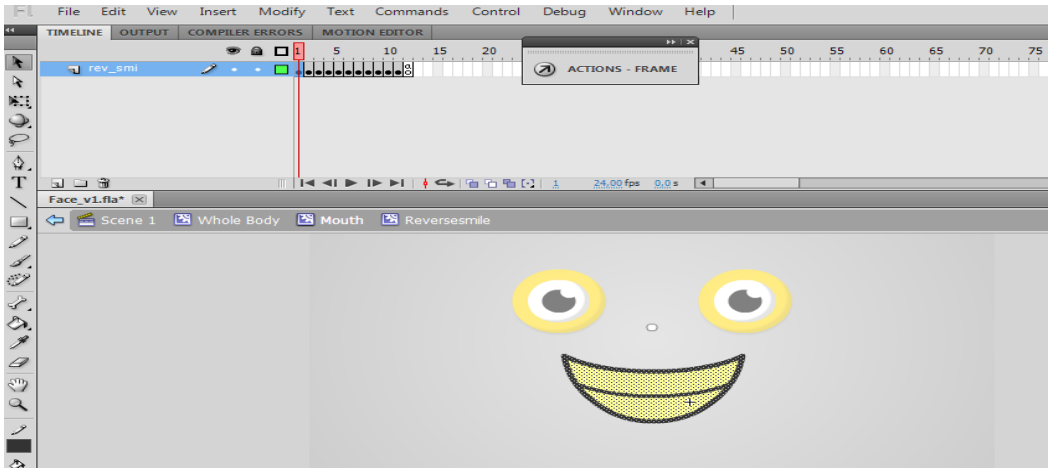


Slika 9. Usta se 'otvaraju' kroz slijed okvira.



Slika 10. Usta u zadnjem okviru, potpuno 'otvorena', čime je animacija završena.

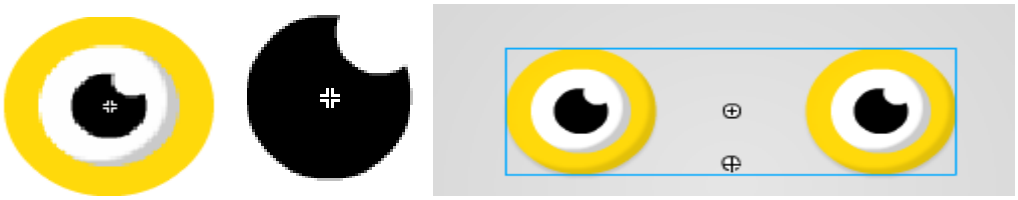
Postupak se ponavlja za sve ostale modele, čime se dobiju četiri animacije 'otvaranja usta'. Postupak se mora ponoviti u obratnom smjeru - tj. da se usta 'zatvore' i nestanu. Postupak je isti kao i za prethodne animacije, samo što su okviri nanizani u obratnom smjeru [Slika 11], tj. početni okvir prikazuje puna usta, koja se 'smanjuju' te na kraju 'nestanu'.



Slika 11. Raspored okvira za animaciju 'zatvaranja usta' - sve ćelije su obrnute.

3.3. Izrada modela i animacija oči

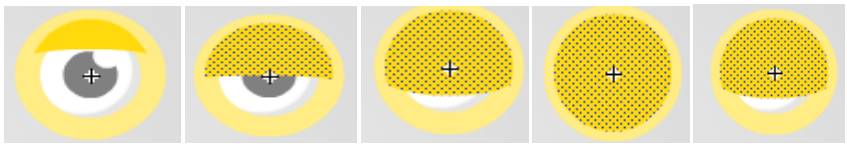
Za izradu oči je bilo potrebno napraviti dva objekta - jedan je bio 'očna duplja', tj. *eye socket*. Drugi je bio sama zjenica, tj. *pupil*. Ta dva elementa potom čine skupni objekt, koji se direktno kopira da napravi par oči [Slika 12].



Slika 12. Modeli - očna duplja, zjenica, te oči.

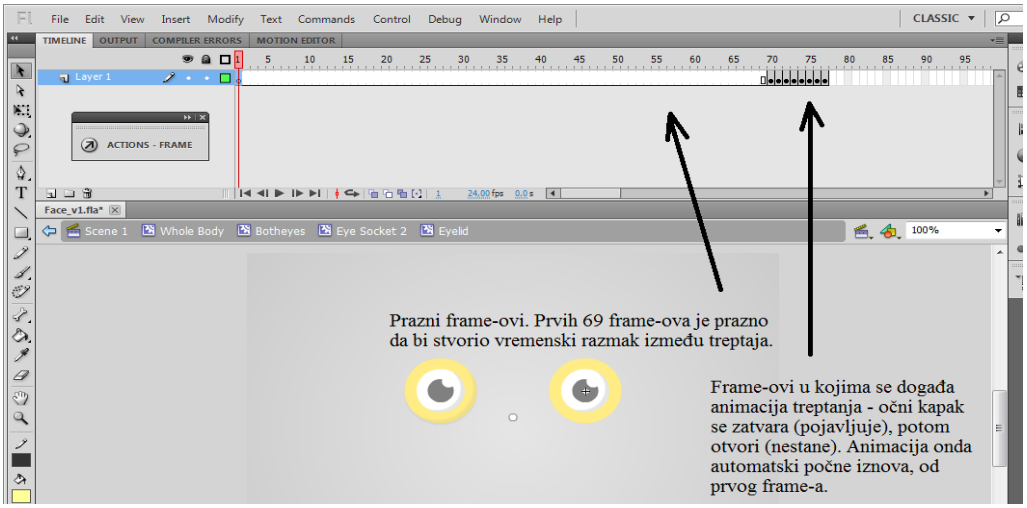
Za animaciju pomicanja oči nije bilo nužno izraditi raspored kao kod usta, jer je to kasnije urađeno preko programskog koda, koristeći '*tween*' naredbe.

Da oči budu potpuno realistične, trebaju biti u stanju 'žmirkati'. To se postiže tako da se izradi dodatni model - žuti krug koji prekriva bijeli dio oka i koji se ponaša kao očni kapak [Slika 13].



Slika 13. Animiranje treptaja oči – modeli.

Potom se, koristeći istu metodu kao kod usta, izradi raspored okvira kod kojega se taj žuti krug 'smanjuje', čime imitira treptanje oči [Slika 14].



Slika 14. Raspored okvira za animaciju očnog kapka.

Animaciju treptanja ne treba daljnje programirati - ona se automatski pokrene na početku pokretanja programa, te se beskonačno ponavlja. Time se stvara dojam da oči periodično trepću.

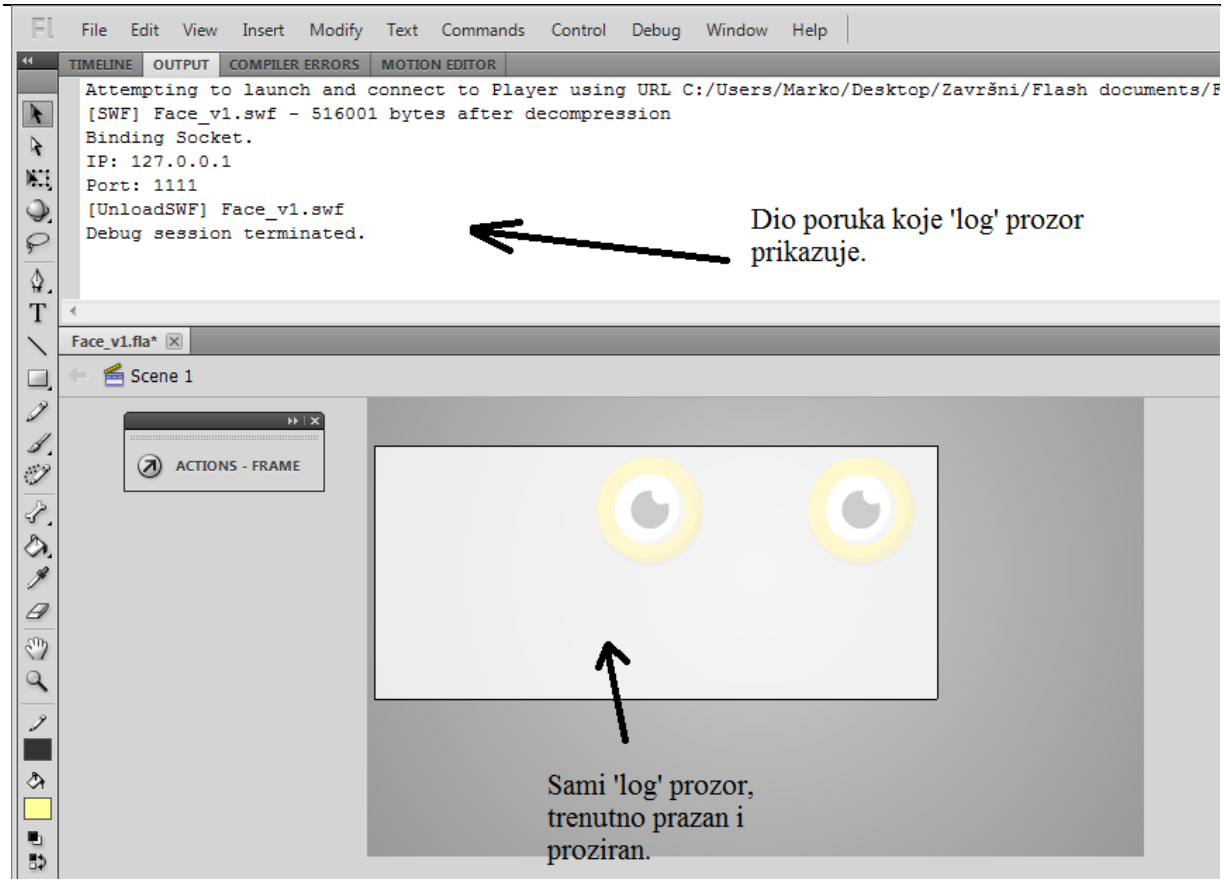
3.4. Prozor za obavijesti *(Log)*

Kad se govori o *log* prozoru, misli se na prozor koji prati i bilježi sve automatske poruke koje program generira, tj. prozor za obavijesti. Neke od njih (poput poruka vezanih uz grešku pri programiranju ili testiranju programa) su dio Flash Professional alata, dok su druge programirane baš za ovaj program.

Sve te poruke su jedino vidljive unutar samih alata u kojima se programira [Slika 15]. Prozor za obavijesti služi da bi se te poruke prenijele u pokretanje samog programa, tako da korisnik ne treba alate za programiranje da bi odredio stanje programa. To je naročito korisno pri uspostavi mrežne veze, koja sama po sebi ne daje nikakve naznake da je došlo do njenog uspostavljanja.

Za izradu *log* prozora, korišteni su grafički alati da se napravi jednostavan, proziran prozor (nazvan '*logbox*'). Unutar tog prozora se kasnije smješta, koristeći programski kod, prozor s tekstom (*text-box)* koji ispisuje poruke programa.

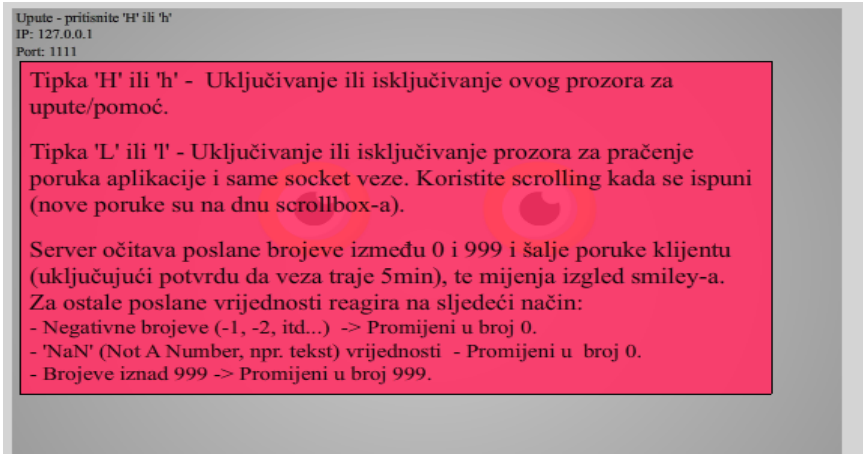
Da prozor ne smeta korisniku, on je na početku programiran da bude 'nevidljiv', svaki puta kada se pokrene program (tzv. *visibility toggle*). On se potom može, pritiskom na tipku 'l' ili 'L', prikazati, ili ponovno sakriti. Programski kod kojim se to postiže će se detaljno razraditi u poglavljima koja slijede.



Slika 15. Prozor za obavijesti (*Log* prozor), kada je vidljiv.

3.5. Prozor za pomoć *(Help)*

Help prozor je u biti prozor u kojemu se nalaze upute za korištenje programa [Slika 16]. Kao i *log* prozor, programiran je da na početku bude skriven, no može se pozvati ili ponovno ukloniti pritiskom na tipku 'h' ili 'H'. Sam prozor je inače statičan i nepromjenjiv, kao što su i upute u gornjem lijevom kutu programa.



Slika 16. Prozor za pomoć (*Help* prozor), kada je vidljiv.

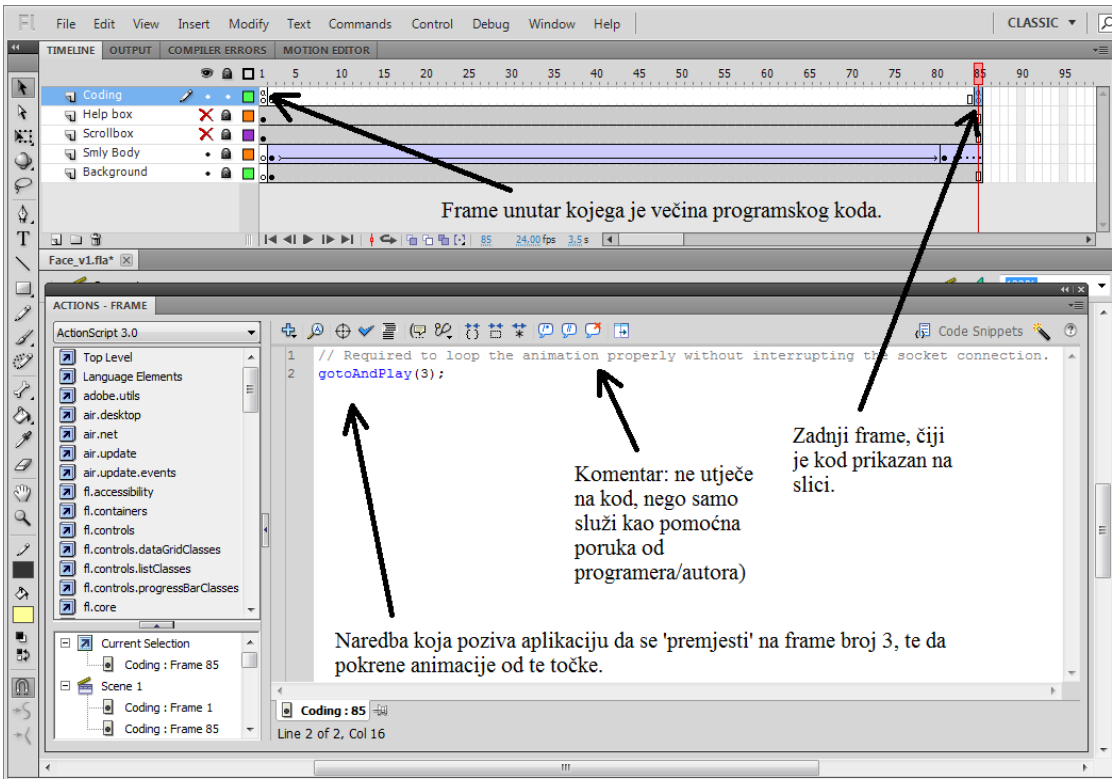
4. IZRADA PROGRAMSKOG KODA

Preostalo je objasniti '*coding*', sloj unutar kojega se nalazi većina programskog koda.

4.1. Podjela programskog koda unutar rasporeda okvira

Zbog ograničenja radnog okruženja, nužno je bilo da se većina programskog koda u programu stavi u prvi, početni okvir. To osigurava da se taj programski kod odmah pokrene, u prvom trenutku pokretanja programa. Tu se još mora napomenuti da se ostali slojevi u biti ne prikazuju u tom početnom okviru, nego tek u drugome. Razlog tome jest što inače dolazi do programskog konflikta između animacija koje se automatski pokrenu kada dođe do tog momenta, te programskog koda koji daje daljnje naredbe tim istim animacijama.

Također je potrebno staviti jednostavnu naredbu '*gotoAndPlay(3);*' na zadnji okvir programa [Slika 17]. Ta naredba osigurava da se animacije pravilno ponavljaju, tj. da ne dođe do stanja gdje se animacije lica 'zamrznu', i time prouzroče daljnje greške.



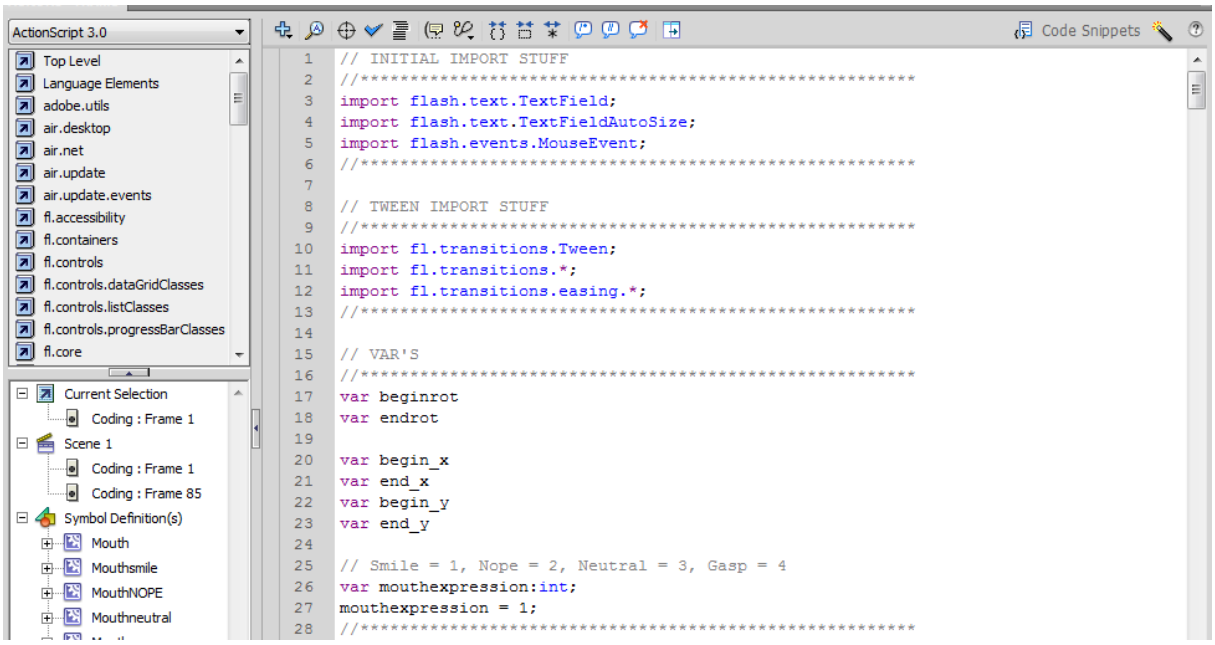
Slika 17. 'Coding' sloj: Pregled prozora za kodiranje, kod zadnjeg sloja, te pojašnjenje.

4.2. Programski kod prvog okvira

Glavni programski kod je podijeljen na nekoliko skupina, te pod-skupina unutar istih: Uvođenje programskih naredbi i varijabli; naredbe za upravljanjem nad *Help* i *Log* prozorima; programski kod koji utvrđuje i upravlja mrežnom vezom (te sva pravila i naredbe vezane uz isti); upravljanje animacijama preko matematičkih funkcija; te dio za identifikaciju i prikaz grešaka koje se mogu javiti.

4.2.1. Inicijalizacija programskog koda i uspostavljanje početnih varijabli

Da bi mogli koristiti određene dijelove programskog koda, treba se prvo 'uvesti' ili 'pozvati' (*import*) određene dijelove programskog koda, tj. učitati njihove naredbe [Slika 18], da bi ih program mogao prepoznati kao specifične naredbe (umjesto običnog teksta). Na slici su prikazane one korištene za *tween* animacije, te prozore s tekstom.



Slika 18. Pozivanje *import* naredbi, te varijabli.

Pritom se još uspostavljaju i početne varijable [Slika 18], a to su vrijednosti koje će programske naredbe biti u stanju raspoznati kao određen oblik informacije (tekst, broj, binarna vrijednost, itd.), te ovisno o vrsti, koristiti za interakcije. Varijable koje su potrebne za programski kod su:

- ***beginrot, endrot;*** korištene pri izračunu kuta rotacije, za animaciju 'okretanja' lica.

- ***begin_x, end_x, begin_y, end_y*;** korištene za izračun početne i krajnje prostorne koordinate zjenica oči.
- ***mouthexpression:int;*** oblika broja (*int, integer*), korišten za određivanje trenutnog stanja (tj. modela) usta. Brojevi 1, 2, 3 i 4 simboliziraju svaki od četiri modela lica. Pošto program uvijek započinje s modelom/animacijom smiješka, varijablu se početno odredi kao broj 1 (*mouthexpression = 1;*).

4.2.2. Naredbe za prozor za obavijesti (*Log*)

Programski kod registra informacija [Slika 19] služi za sljedeće funkcije:

Redovi 33-38: Uspostavljanje i stvaranja samog 'prozora' unutar kojega će se nalaziti tekst, za što se koriste komande koje stvaraju varijablu od sadržaja prozora i objekt od samog prozora, pri čemu se određuje visina i širina, te pravila automatskog oblikovanja teksta.

Red 41: Određuje se da je prozor u 'nevidljivom' stanju, da ne smeta i zaklanja ostale objekte programa kada nije potrebno.

Red 44: Funkcija koja ispisuje poruke od alata za programiranje, tj. automatske sistemske poruke ili obavijesti (za koje je inače potrebno imati alate za programiranje da bi bile vidljive).

Redovi 47-56: Naredbe koje određuju *Event*, tj. događaj, te reakciju na isti. Događaj koji se prati jest pritisak tipki 'l' ili 'L', a reakcija jest da se mijenja stanje 'vidljivosti' prozora.

```
// LOGBOX
//*********************************************************
// Variable and settings for the text field, to keep a log of the connection code.
var tf:TextField = new TextField();
tf.wordWrap = true;
tf.width = 300;
tf.height = 180;
tf.autoSize = TextFieldAutoSize.NONE;
logbox.addChild(tf);

// This bit to make the log box initially 'invisible'
logbox.visible = !logbox.visible;

// Function to keep a log of whatever is inputted into compiler.
function log(str) { tf.appendText("\n" + str); }

// This is the code to show the log screen when you press 'l'.
stage.addEventListener(KeyboardEvent.KEY_DOWN,logboxhandler);
function logboxhandler(event:KeyboardEvent){
    if(event.keyCode == 76 || event.charCode == Keyboard.L){
        // Sets the visibility of it to the opposite of what it is (starts off).
        logbox.visible = !logbox.visible;

        // Enable mouse events/selection based on visibility (it's all just booleans)
        logbox.enable = logbox.visible;
    }
}
//*********************************************************
```

Slika 19. Programski kod - Prozor za obavijesti (*Log*).

4.2.3. Naredbe za prozor za pomoć (*Help*)

Jedine programske naredbe koje su potrebne za upute [Slika 20] jesu iste koje koristi registar, tj. one za određivanje 'vidljivosti' prozora, za što se prate tipke 'h' i 'H' na tipkovnici.

```
// HELPBOX
//*********************************************************
helpscreen.visible = !helpscreen.visible;

stage.addEventListener(KeyboardEvent.KEY_DOWN,helpboxhandler);
function helpboxhandler(event:KeyboardEvent){
    if(event.keyCode == 72 || event.keyCode == Keyboard.H){
         // Sets the visibility of it to the opposite of what it is (starts off).
        helpscreen.visible = !helpscreen.visible;

         // Enable mouse events/selection based on visibility (it's all just booleans)
        helpscreen.enable = helpscreen.visible;
    }
}

//*********************************************************

```

Slika 20. Programski kod - Upute (*Help*).

4.2.4. Uspostava bežične mrežne veze

Kao i prije, potrebno je učitati programske naredbe [Slika 21], te uvesti varijable. Ovaj put, one su:

- *serverSocket*; varijabla za naredbe izvršitelja veze (tj. samog programa).
- *clientSockets*; varijabla za naredbe vezane uz korisnika veze.
- *ServerSocketSmiley*; ime programske funkcije koja određuje IP i *port* adresu servera, te 'preuzima' iste.

```
77
78  //SOCKET STUFF
79  //**************************************************************
80  // More import stuff...
81  import flash.events.SecurityErrorEvent;
82  import flash.system.Security;
83  import flash.display.Sprite;
84  import flash.events.Event;
85  import flash.events.IOErrorEvent;
86  import flash.events.ProgressEvent;
87  import flash.events.ServerSocketConnectEvent;
88  import flash.net.ServerSocket;
89  import flash.net.Socket;
90  import flash.utils.Timer;
91  import flash.events.TimerEvent;
92
93  // global vars...
94  var serverSocket:ServerSocket;
95  var clientSockets:Array = new Array();
96
97  ////////////////////////////
98  ServerSocketSmiley();
99  ////////////////////////////
```

Slika 21. Uvođenje programskih naredbi i varijabli za mrežnu vezu.

Za funkciju uspostave servera, *ServerSocketSmiley*, [Slika 22] je potrebno:

- Stvoriti 'server' kao programski objekt koji se može pratiti (red 106).
- Pridodati serveru mogućnost praćenja događaja vezanih uz uspostavljanje i zatvaranje mrežne veze (red 109-110). To će omogućiti automatsko slanje poruka korisniku, pri uspostavljanju veze.
- Uvesti serversku lokaciju - IP adresu (127.0.0.1, lokalna adresa), broj *port-a* (1111), te 'preuzeti' iste na računalu na kojemu će se program pokretati (red 113).
- Dodati naredbe kojima se većina događaja mrežne veze može pratiti i prikazati preko prozora za obavijesti (red 115-124). Bez ovih naredbi, skoro je nemoguće odrediti u kojem je stanju mrežna veza.
- Dati programu naredbu da 'osluškuje' postojeće adrese na računalu, za korisnika koji se pokušava spojiti na program (red 123).

```
101 function ServerSocketSmiley()
102 {
103     try
104     {
105         // Create the server socket.
106         serverSocket = new ServerSocket();
107 
108         // Add the event listener
109         serverSocket.addEventListener( Event.CONNECT, connectHandler );
110         serverSocket.addEventListener( Event.CLOSE, onClose );
111 
112         // Bind to the port and IP variable.
113         serverSocket.bind(1111, "127.0.0.1");
114 
115     trace ("Binding Socket.");
116     log ("Binding Socket.");
117     trace ("IP: " + serverSocket.localAddress);
118     log ("IP: " + serverSocket.localAddress);
119     trace ("Port: " + serverSocket.localPort);
120     log ("Port: " + serverSocket.localPort);
121 
122         // Listen for connections.
123         serverSocket.listen();
124         log( "Listening on port " + serverSocket.localPort +"...");
125 
126     }
127     catch(e:SecurityError)
128     {
129         log(e);
130     }
```

Slika 22. Funkcija za naredbe servera.

Sljedeća funkcija jest ona koja određuje ponašanje programa kada se mrežna veza uspješno uspostavi, funkcija imena *connecthandler* [Slika 23]. Unutar nje se za korisnika uspostavi programski objekt (imena *socket*), te se korisniku i samom serveru automatski pošalje tekstualna poruka (r. 144 i 149) o uspješno uspostavljenoj vezi. Naredba '*flush*' služi da se mrežna veza 'pročisti' od neželjenih i zaostalih informacija koje bi mogle dovesti do grešaka.

```
133 function connectHandler(event:ServerSocketConnectEvent):void
134 {
135     //The socket is provided by the event object.
136     var socket:Socket = event.socket as Socket;
137     clientSockets.push( socket );
138 
139     socket.addEventListener( ProgressEvent.SOCKET_DATA, socketDataHandler);
140     socket.addEventListener( Event.CLOSE, onClientClose );
141     socket.addEventListener( IOErrorEvent.IO_ERROR, onIOError );
142 
143     //Send a connect message.
144     socket.writeUTFBytes("| Connection established to server: "+serverSocket.localAddress +", port "+serverSocket.localPort +". |");
145 
146     // IMPORTANT to flush! <-----------------
147     socket.flush();
148 
149     log("Sending connect message ('Connection established.')");
150 
151     // Timer ("t") that sends a message after 5 minutes elapsed, one time...
152 
153     var t:Timer = new Timer(300000, 1);
154     t.addEventListener(TimerEvent.TIMER, handleTimer);
155     t.start();
156 }
```

Slika 23. Funkcija naredbi za uspješno uspostavljenu mrežnu vezu, i uspostavu štoperice.

Za svrhu testiranja održivosti veze, još se zadaje i štoperica od pet minuta (u obliku od 300,000 okvira). Ona odbrojava vrijeme/vremenske okvire od početka veze, te nakon pet minuta šalje automatske poruke serveru i korisniku da se potvrdi trajanje veze [Slika 24].

```
157
158 // .. every 5 minutes after the connection is established, write to the socket.
159 function handleTimer(e:TimerEvent):void
160 {
161     // Makes a log prior to sending message to client...
162     log("LOG: 5 minutes have elapsed on the server. Sending message to client.");
163
164     // The actual message sent.
165      clientSockets[0].writeUTFBytes("| TIMER MESSAGE: 5 minutes have elapsed since connection to port "+ serverSocket.localPort +". |");
166      clientSockets[0].flush();
167 }
168
```

Slika 24. Funkcija štoperice.

4.2.5. Naredbe za reakcije (animacije) na poslane podatke tijekom veze.

Sljedeća funkcija, *socketDataHandler*, jest ujedno i najveća. U njoj se nalaze programske naredbe koje primaju podatke koje korisnik (tj. odabrani robot) šalje programu, pretvara iste u pravilan numerički oblik, te ovisno o dobivenoj vrijednosti, šalje naredbe programu da reagira tako da utječe na animacije prikazanog lica.

4.2.5.1. Naredbe za inicijalnu identifikaciju i transformaciju poslanih podataka…

Za robota se pretpostavlja da je u stanju poslati podatke u obliku brojeva od 0-999, te da je program programiran da reagira na te vrijednosti. U slučaju da robot šalje druge vrste podatka, poput teksta, teksta i brojeva, negativnih brojeva, brojeva većih od 999, ili bilo koje ne-brojčane vrijednosti (*Not A Number / NaN*), te vrijednosti se automatski mijenjaju u brojčanu vrijednost nula ili 999.

Da bi program pravilno prepoznao vrstu podatka, najprije mora znati kada dolazi do slanja istih [Slika 25]. To radi tako da prati broj dostupnih bajtova (*available bytes*), tj. memorijsku vrijednost poslanih podataka. Ukoliko je vrijednost dostupnih bajtova veća od nule, to znači da su poslani podaci, te ih program očitava i bilježi u registru (r. 173-180).

```
169 function socketDataHandler(event:ProgressEvent):void
170 {
171     var socket:Socket = event.target as Socket;
172 
173     trace(socket.bytesAvailable + ' byte(s) available');
174     log(socket.bytesAvailable + ' byte(s) available');
175     // Read the socket data for any positive bytes being sent...
176     if (socket.bytesAvailable > 0)
177     {
178     //Reads the message from the socket...
179     var msg:String = socket.readUTFBytes( socket.bytesAvailable );
180     log( "Received: " + msg);
181 
182         // Creates a number variable, "sentnumber", which the input box layer will use...
183         // ... to alter the smiley.
184         var sentnumber:Number = Number(msg);
185 
186         // Tracing for a NaN (Not A Number) value from the sent data.
187         trace("Is the sent value a NaN? Response: " + isNaN(sentnumber))
188         log("Is the sent value a NaN? Response: " + isNaN(sentnumber))
189 
190         // If there is a NaN value sent, it is converted into the number zero instead.
191         if (isNaN(sentnumber) == true)
192         {
193             sentnumber = 0
194             trace ("Sent value was NaN, converting to zero.");
195             log ("Sent value was NaN, converting to zero.");
196             trace ("Converted to number: " + sentnumber);
197             log ("Converted to number: " + sentnumber);
198         }
```

Slika 25. Pretvorba poslanih podataka.

Program pritom uvodi varijablu imena *msg* (r. 179). Ta varijabla, koja je na početku tekstualnog (*string*) oblika, izjednačuje se s vrijednosti varijable *sentnumber* (r. 184), da bi se međusobno logički usporedile.

Varijabla *sentnumber* se uspoređuje po sljedećim kriterijima, koristeći logičko uspoređivanje [slika 26]:

- *if (isNaN(sentnumber) == true);* naredba koja provjerava je li vrijednost dobivena od poslanih podataka u obliku broja, koristeći binarni upit (*true/false*, tj. istina/laž). Ukoliko nije, vrijednost se automatski izjednačuje s nulom. Ukoliko jest broj, ide sljedeća usporedba.
- *else if (sentnumber > 999);* koja provjerava je li vrijednost broja veća od 999. Ukoliko jest, vrijednost se izjednačuje s brojem 999.
- *else if (sentnumber < 0);* koja provjerava je li broj manji od nule (negativni). Ukoliko jest, izjednačuje se s brojem nula (0).

Automatske poruke ujedno obavještavaju korisnika programa o pretvorbi vrijednosti. Slijed tako uvijek završava s brojem između 0-999, bez obzira na oblik poslanih podataka.

```
190         // If there is a NaN value sent, it is converted into the number zero instead.
191         if (isNaN(sentnumber) == true)
192         {
193             sentnumber = 0
194             trace ("Sent value was NaN, converting to zero.");
195             log ("Sent value was NaN, converting to zero.");
196             trace ("Converted to number: " + sentnumber);
197             log ("Converted to number: " + sentnumber);
198         }
199         // If a value higher than 3 digits is sent, it's converted to 999.
200         else if (sentnumber > 999)
201         {
202             sentnumber = 999
203             trace ("Sent value was too high, converting to maximum allowed.");
204             log ("Sent value was too high, converting to maximum allowed.");
205             trace ("Converted to number: " + sentnumber);
206             log ("Converted to number: " + sentnumber);
207         }
208         // Negative numbers are converted to zero as well.
209         else if (sentnumber < 0)
210         {
211             sentnumber = 0
212             trace ("Sent value was negative, converting to zero.");
213             log ("Sent value was negative, converting to zero.");
214             trace ("Converted to number: " + sentnumber);
215             log ("Converted to number: " + sentnumber);
216         }
217         // Otherwise, the 0-999 number is used to alter the smiley and digit boxes.
218         else {
219             trace("Converted to number: " + sentnumber);
220             log("Converted to number: " + sentnumber);
221         }
222
223     // 'Echoes' the received message back to the sender...
224     socket.writeUTFBytes("| Echo: " + msg + " |");
```

Slika 26. Logička usporedba, praćenje, te konverzija poslanih podataka.

4.2.5.2. Određivanje jedinica, desetinki, i stotinka unutar broja.

Nakon usporedbe, vrijednost *sentnumber* jest broj između 0-999. Taj broj služi kao referenca za tri različite animacije, ovisno o vrijednosti njegove jedinice, desetinke, ili stotinke:

- Izmjena oblika usta, nizanjem prethodno izrađenih animacija za pojedini model usta, prati stotinku kao referentnu vrijednost.
- Za rotaciju cjelokupnog 'lica' ulijevo ili udesno gleda se desetinka.
- Pomak zjenica unutar očnih duplji, tako da 'gledaju' u određenom smjeru, prati vrijednost jedinice.

Naravno, potrebno je prvo izolirati jedinice, desetinke, i stotinke iz broja. To se izvršava matematičkim formulama na slici [Slika 27]. Prvo se uspostavi varijabla *code*, koja se potom matematički pretvara u traženu vrijednost. Pritom se koristi naredba *Math.floor* da bi dobili vrijednost tog broja zaokruženog na nižu vrijednost.

```
231 // Turns the value from input box into a number.
232       var code:Number = Number(sentnumber);
233
234 // Takes the 1s from the code, and puts it in the 1s box...
235       var jedinica:Number = Math.floor(code) % 10;
236
237 // Takes the 10s from the code, and puts it in the 10s box...
238       var desetka:Number = Math.floor(code/10^0) % 10;
239
240 // Takes the 100s from the code, and puts it in the 100s box...
241       var stotinjka:Number = Math.floor(code/100);
```

Slika 27. Matematičke funkcije za varijable, imena *jedinica, desetka, i stotinjka*. Naredba *Math.floor* pritom daje cijeli broj, zaokružen na nižu vrijednost, umjesto decimalnog.

Varijable imena *jedinica*, *desetka*, i *stotinjka*, numeričkih vrijednosti od 0 do 9, sada služe kao referentne informacije za odabir triju animacija koje se pokrenu kada je podatak poslan.

4.2.5.3. Programska funkcija animacije usta

Za animaciju usta se prati broj stotinki. Kada je broj stotinke vrijednost između 1-3, pokreće se animacija 'pojavljivanja' smiješka. Ukoliko trenutno stanje nije smiješak (tj. neki drugi model usta je prisutan), za to stanje se pokreće obratna (*reverse*), animacija tog modela (npr. za neutralna usta, ona se zatvaraju), te se tek onda, nakon što model 'nestane', pokreće animacija smiješka.

Programske naredbe koje izvršavaju taj slijed animacija [Slika 28] prvo pregledaju poslani broj (da se zna koja usta se trebaju 'otvoriti'), potom provjeravaju trenutno stanje (da bi se odigrala animacija 'zatvaranja' usta), izvršavaju prikladnu animaciju (naredbom *gotoAndStop*, koja koristi imena okvira određenih ranije), uspostavljaju štopericu (za vremenski razmak između animacija), te napokon daju naredbu za izvršenje animacije usta za vrijednost stotinke.

Slika 28. Funkcije naredbi za animaciju usta.

Ostali modeli lica reagiraju na vrijednosti 0 (neutralno), 3-6 (čuđenje), 6-9 (nezadovoljstvo). Tako za devet mogućih brojeva postoji četiri različitih 'konačnih' stanja. Isti programski kod se ponavlja za ostale modele usta, uz prikladne izmjene vrijednosti za animacije i varijable. Time se dobiva animacija koja realistično zatvara i otvara usta, svaki puta kada se pošalje podatak preko mrežne veze.

4.2.5.4. Programska funkcija rotacije lica

Za rotaciju lica se može koristi jednostavna naredba *rot*, koja neki objekt, ovisno o brojčanoj vrijednosti stavljenoj iza iste, rotira za toliko stupnjeva od svog centra. No to nije dovoljno za ovaj zadatak, jer se ta rotacija odvija bez vidljive 'tranzicije'. Da bi rotacija bila realistična (tj. da se lice vidljivo rotira u nekom vremenu), potrebne su tzv. *tween* (od riječi *inbetween*, između) naredbe. Te naredbe djeluju 'između' početka i kraja animacije, tj. njima se zadaju vrijednosti poput početnog stanja, krajnjeg stanja, među-stanja, ponašanja između tih stanja, itd.

Tween je u obliku sljedeći programskog koda:

Tween(ime_objekta_koji_animiramo,"ime",metoda_animairanja,početna_točka ,krajnja_točka, trajanje_animacije,korištenje_sekunda)

Što se tiče realističnog kretanja lica, dovoljno je da u *tween* uvede naredba *easeOut*. Ta naredba govori animaciji da 'krene' pri velikoj brzini, te da se usporava kako se približava krajnjoj točki [Slika 29].

Rotacija lica ovisi o broju desetinke, te ovisno o istome, određuje za koliko stupnjeva (od -20 do +20, s početnom točkom u nuli) se okreće, pri čemu koristi ranije utvrđene varijable *beginrot* i *endrot* kao referentne točke.

```
385 // Again, it first checks the zero...
386 if (desetka == 0) {
387     endrot = 0
388     var rot0:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1, true);
389     beginrot = endrot;
390        }
391        else if (desetka == 1) {
392            endrot = -20
393            var rot10:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1, true);
394            beginrot = endrot;
395         }
396        else if (desetka == 2) {
397            endrot = -15
398            var rot20:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1, true);
399         beginrot = endrot;
400         }
401        else if (desetka == 3) {
402            endrot = -10
403            var rto30:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1, true);
404         beginrot = endrot;
405         }
```

Slika 29. *Tween* naredbe za rotaciju lica. Desetinka određuje broj stupnjeva pomaka od početnog položaja, dok *tween* naredbe pridaju 'realističnom' kretanju pri rotaciji. Na slici su prikazana tri stanja, za desetke od 1 do 3.

4.2.5.5. Programska funkcija pomaka oči

Oči kao referencu za pomak koriste vrijednost jedinica, te ujedno koriste *tween* naredbe da bi animacija izgledala realistično, slično kao i za rotaciju lica [Slika 30]. Glavna razlika jest u tome što se za oči ne koriste naredbe za rotaciju, nego se oči direktno pomiču prema koordinatama x i y, uzevši sredinu očne duplje kao nultu točku (0,0). Za svako stanje oči bilo je potrebno napraviti četiri *tween*-a - dva za svako oko, za određivanje koordinata x i y. Pritom ranije uspostavljene naredbe *begin_x*, *end_x*, *begin_y*, i *end_y* služe kao referenca trenutne i konačne pozicije zjenice unutar očne duplje, te se logički izjednačavaju (tj. pamti se pozicija oči nakon pomaka, te se uzima u obzir za daljnje pomake).

```
437 // Checks the 1's numbers, and moves the eyes of the smiley for each.
438 if (jedinica == 0) {end_x = 0
439     end_y = 0
440     var eye_x0:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
441     var eye_y0:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
442     var eye2_x0:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
443     var eye2_y0:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
444     begin_x = end_x
445     begin_y = end_y
446 
447        }
448        else if (jedinica == 1) {
449 
450            end_x = 5
451     end_y = 0
452     var eye_x1:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
453     var eye_y1:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
454     var eye2_x1:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
455     var eye2_y1:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
456     begin_x = end_x
457     begin_y = end_y
458 
459        }
460        else if (jedinica == 2) {
461            end_x = 4
```

Slika 30. Funkcije i *tween*-ovi za pomak oči. 2x2 varijabli za x i y koordinate oči.

4.2.6. Završne naredbe

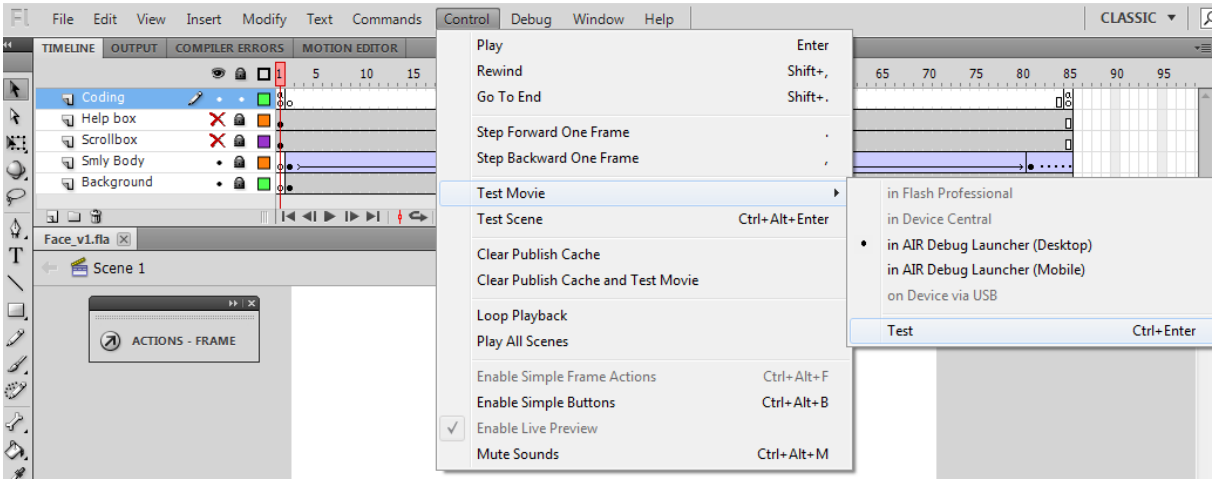
Preostale su još samo završne naredbe [Slika 31], koje uključuju dvije naredbe za bilježenje poruka kada se mrežna veza s korisnikom ili sa serverom zatvori (r. 546-550, i 559-562), te naredba koja otkriva greške do kojih može doći pri pokretanju (r. 553-556).

```
540 }
541 
542 // Error events, helpful to track errors during programming...
543 
544 
545 // Event handler for the client closing.
546 function onClientClose( event:Event ):void
547 {
548     log("Connection to client closed.");
549     //Should also remove from clientSockets array...
550 }
551 
552 // Event handler for errors.
553 function onIOError( errorEvent:IOErrorEvent ):void
554 {
555     log("IOError: " + errorEvent.text);
556 }
557 
558 // Event handler for server closing.
559 function onClose( event:Event ):void
560 {
561     log("Server socket closed by OS.");
562 }
```

Slika 31. Zadnje naredbe programa - funkcija za greške, te prekid veze sa serverom/korisnikom.

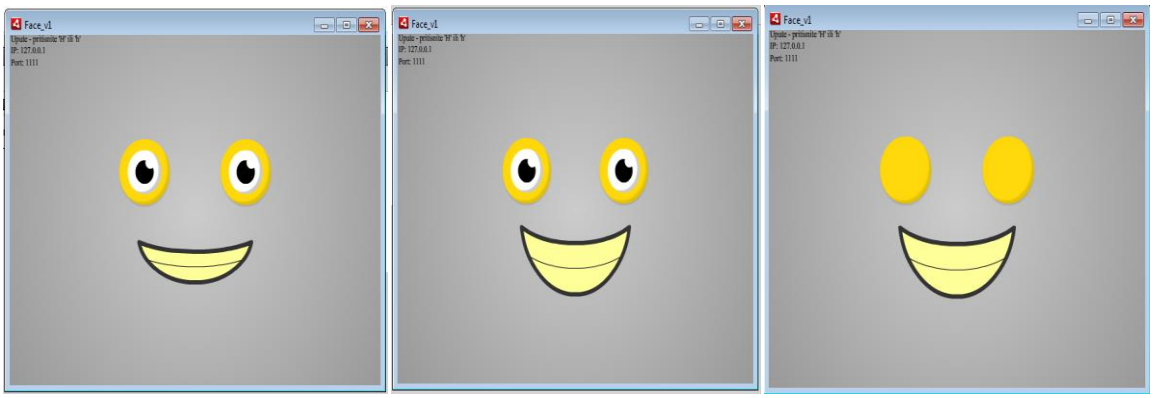
5. TESTIRANJE MREŽNE VEZE I SPREMANJE PROGRAMA

Nakon izrade svih modela, animacija, te programskog koda, preostaje testiranje programa. Program se može direktno testirati s dostupnim alatima [Slika 32].



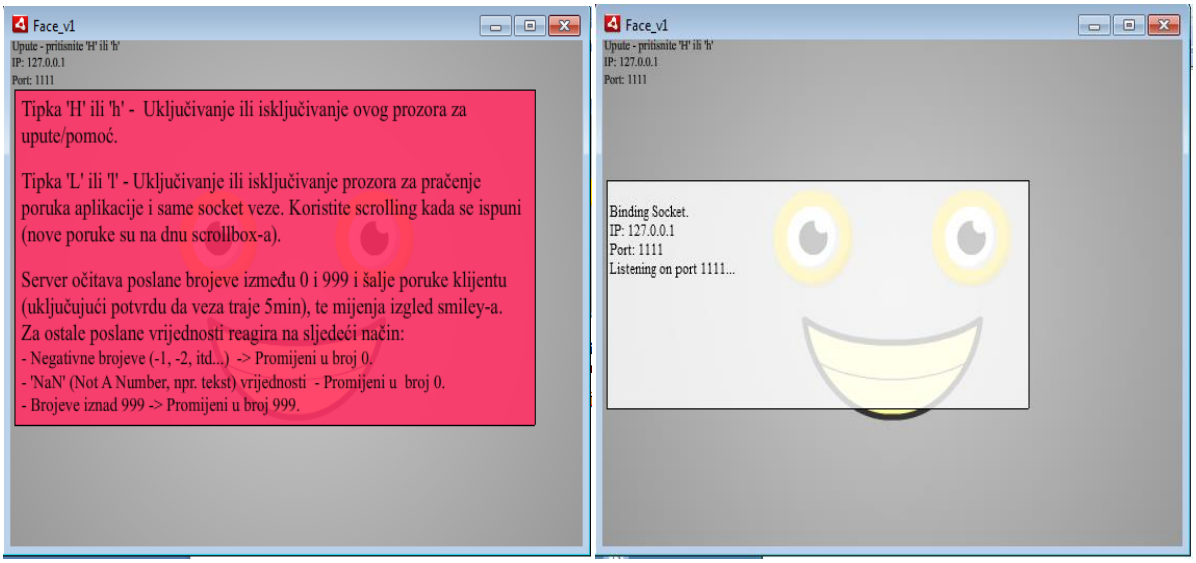
Slika 32. Testiranje je moguće korištenjem Flash Professional Alata.

Nakon pokretanja, otvara se prozor, gdje se prikazuju svi vidljivi slojevi, te se pokreće kratka animacija otvaranja usta [Slika 33]. Oči neprekidno ponavljaju animaciju treptanja svakih nekoliko sekundi.



Slika 33. Program pri pokretanju - usta se animiraju pri pokretanju, a oči 'žmire' svakih nekoliko sekundi.

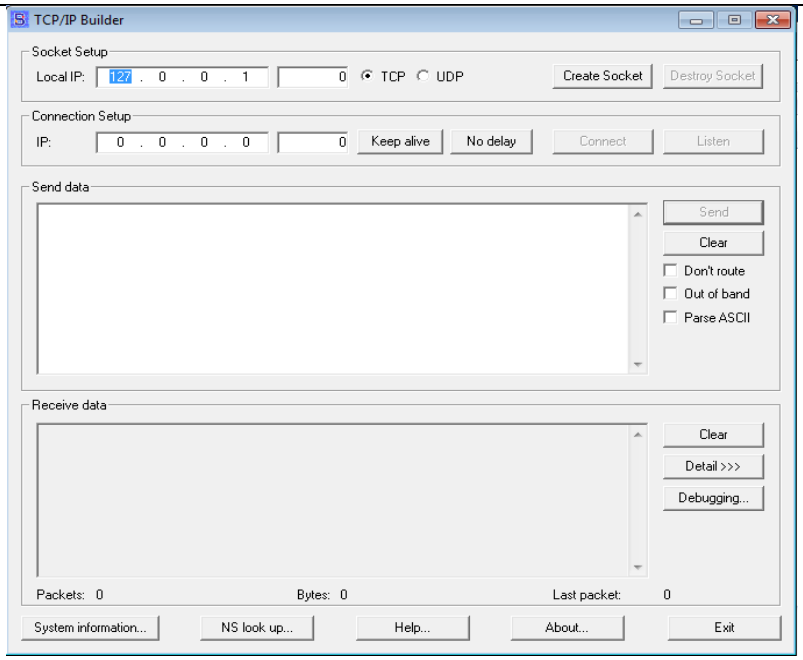
Pritiskom na l, L, h, ili H tipke otvaraju se i zatvaraju skriveni prozori [Slika 34]. Za testiranje mrežne veze bit će potreban *log* prozor, na kojemu se već pri pokretanju može vidjeti da je program uspostavio server, na ispisanoj IP i *port* adresi, te da 'osluškuje' na istom.



Slika 34. Prozori za pomoć i obavijesti (*Help* i *Log* prozor), te ispis poruka.

5.1. TCP/IP Builder - Program za testiranje

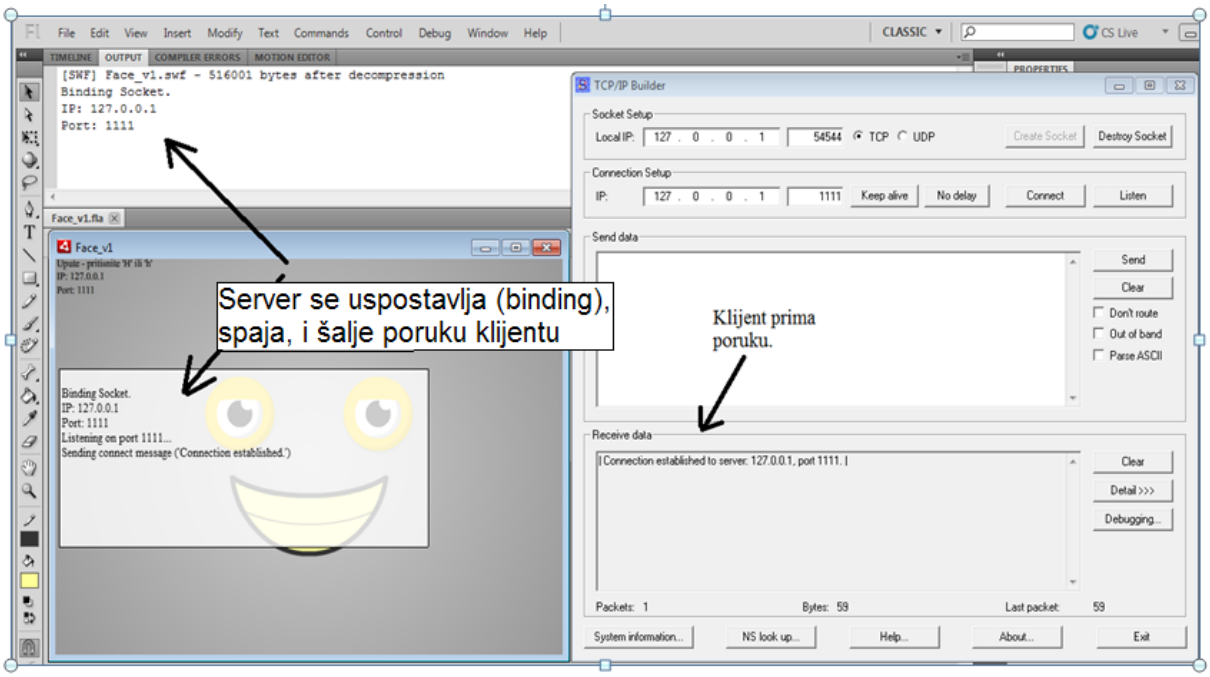
Da bi se pravilno testirao program, potreban je 'korisnik', koji je u stanju spojiti se na poslužiteljski program, te slati i primati podatke. Za tu svrhu koristi se TCP/IP Builder [6], program koja simulira mrežne veze [Slika 35]. U stanju je primati i slati tekstualne podatke preko jednostavnog sučelja, te može uspostaviti vezu ili kao poslužitelj, ili kao korisnik, te je po načinu povezivanja i slanju podataka najsličniji robotu. Za ovaj slučaj će progam biti namješten da se ponaša kao korisnik, koji će se spojiti na izrađeni server program.



Slika 35. Prikaz sučelja TCP/IP Builder-a

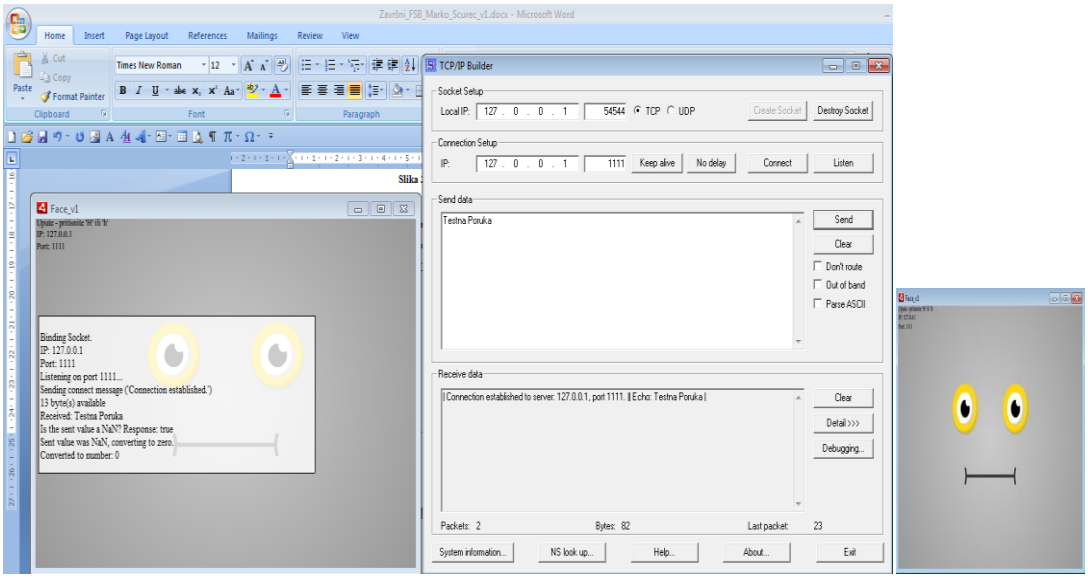
5.2. Uspostavljanje veze

Nakon pokretanja TCP/IP Builder-a, te podešavanja na pravilnu IP adresu i *port* broj, kao korisnika, naređuje mu se da se pokuša spojiti (*connect*). Dokaz uspješnog spajanja se može vidjeti na strani korisnika i servera, preko automatskih poruka koju oba programa ispišu pri spajanju [Slika 36].

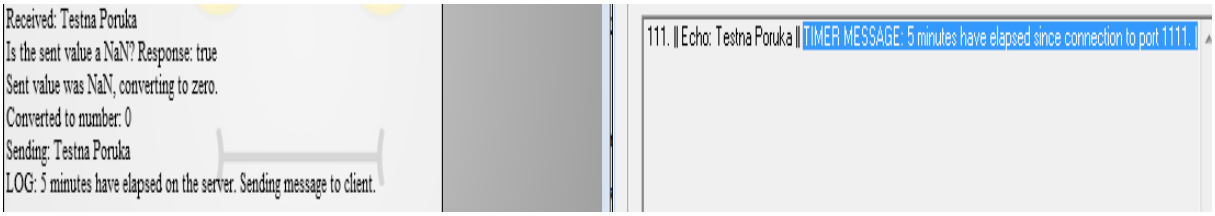


Slika 36. Uspostava veze poslužitelj-korisnik.

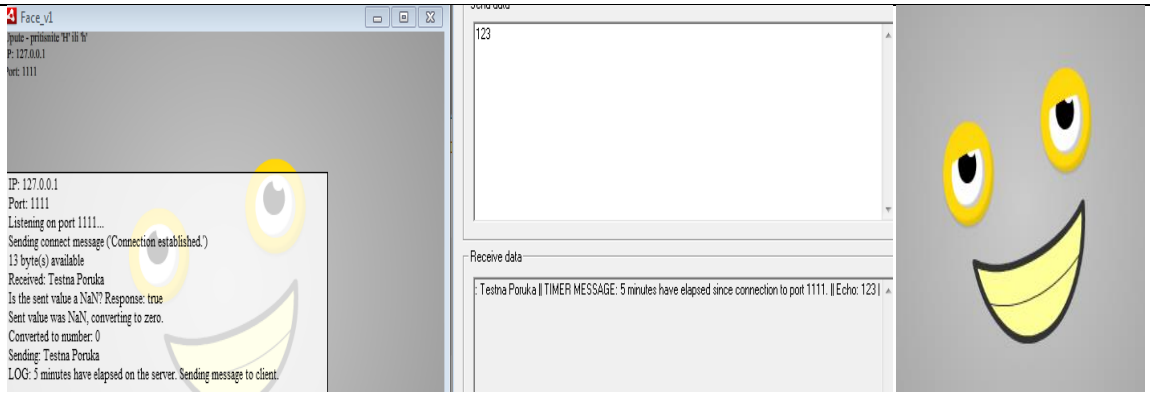
Sada se može testirati slanje podataka. Unutar *Send data* prozora u TCP/IP Builder-u se unese nekoliko poruka. Program reagira na njih s pratećim porukama, pretvorbom podataka (ukoliko nije broj ili vrijednost između 0-999), te s reakcijom lica [Slika 37][Slika 38]. Program nakon pet minuta također šalje informaciju o trajanju mrežne veze, čime se potvrđuje da je veza stabilna [Slika 39]. Isti postupak se nastavlja da se utvrdi pravilno animiranje lika.



Slika 37. Izmjena podataka: Tekst - Vrijednost se pretvara u nulu, te lice mijenja usta.



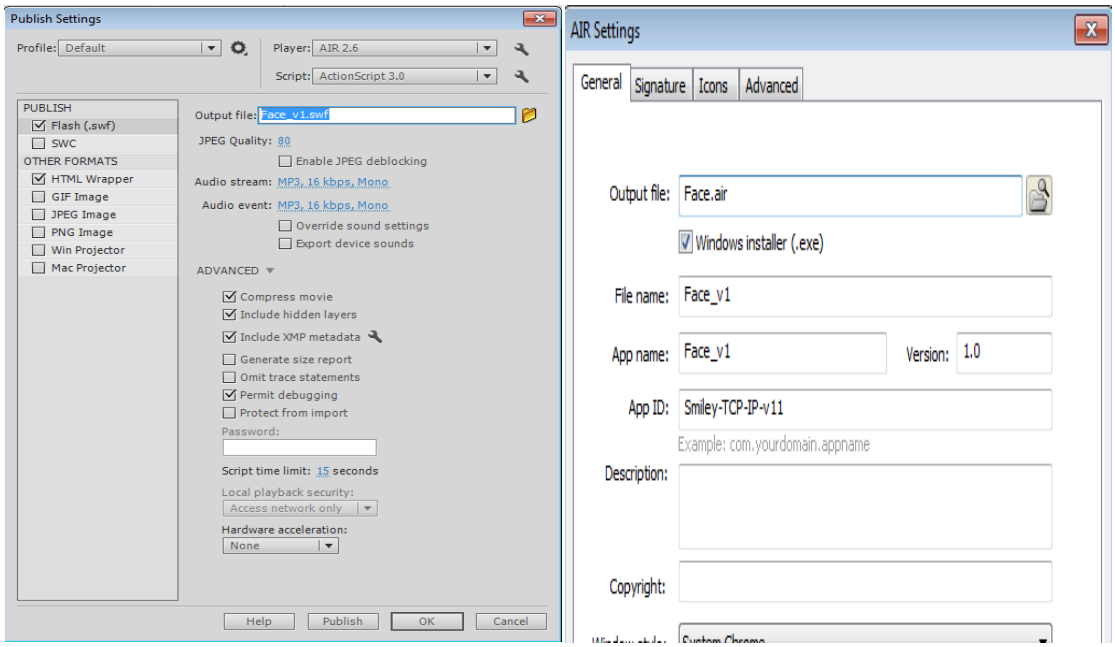
Slika 38. Štoperica od pet minuta automatski šalje poruku o trajanju veze.



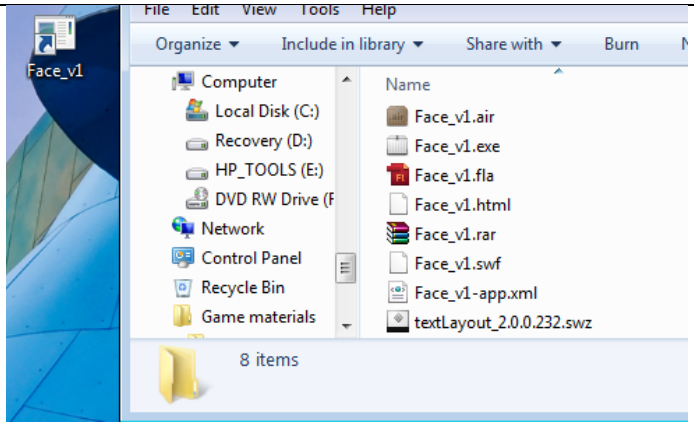
Slika 39. Izmjena podataka: Broj 123 - Vrijednost se uzima kakva jest, lice mijenja rotaciju i smjer oči, te 'vraća' smiješak iz 'neutralnog' stanja usta.

5.3. Spremanje programa

Nakon uspješnog testiranja, preostalo je još samo cijeli program spremiti u pravilnu programsku verziju. To se postiže tako da se sve što je dosad izrađeno objavi (*publish*) u oblik Adobe AIR programa. Pri tom procesu još se odabire da program bude spremljen u obliku *Windows installer-a*, tj. da bude zapakiran na način da se pri pokretanju na računalu automatski instalira, zajedno s Adobe AIR elementima koji su programu potrebni za rad [Slika 40][Slika 41].



Slika 40. *Publish* naredbe, kojima od programa radimo Windows instalacijski alat.



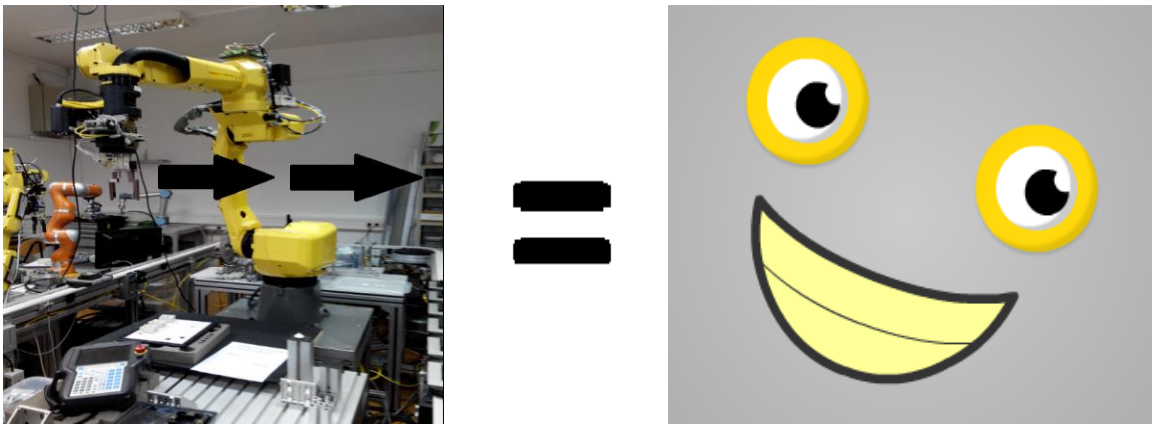
Slika 41. Sve verzije istog programa, dobivene pri spremanju, te program nakon instalacije.

6. PRAKTIČNA PRIMJENA I POPIS VIZUALNIH STANJA

6.1. Praktični primjer - laboratorijski robot

Kao primjer funkcionalnosti, program je sada moguće spojiti s robotom iz Laboratorija za Robotiku, te prikazati stanje ili radnje istog [Slika 42].

Robot se spaja na program preko mrežne veze s drugim računalom (ili istim računalom kojim se trenutno upravlja). Robot je u stanju pomicati se lijevo-desno, stoga kao primjer prikaza stanja, može se programirati program (ili robota - ovisno o tome koliko je praktična izmjena ili prilagodba programskih naredbi koji isti robot šalje programu) da reagira na pomak u specifičnom smjeru s odgovarajućim pomakom na 'licu' programa. Tako se vrlo jednostavno može vidjeti što robot radi preko istog 'lica', te čak prepoznati kada se ponašanje robota ne slaže s onim koje programski kod interpretira, ili u slučaju greške s računalnim sustavom izraditi specifični 'izraz lica' koji služi kao upozorenje ili vizualna poruka korisniku.

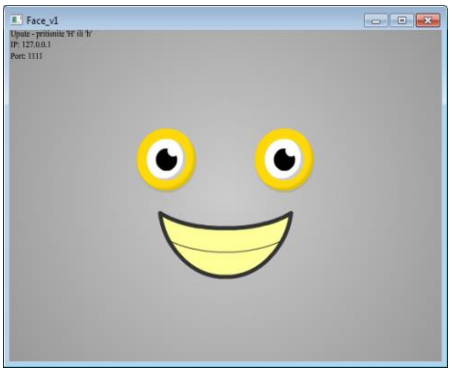
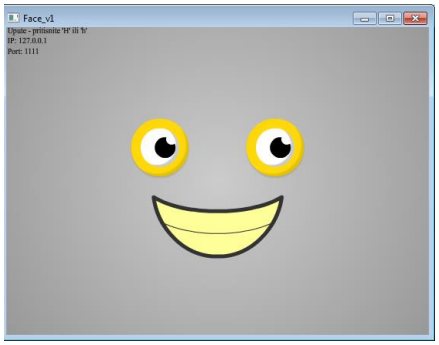
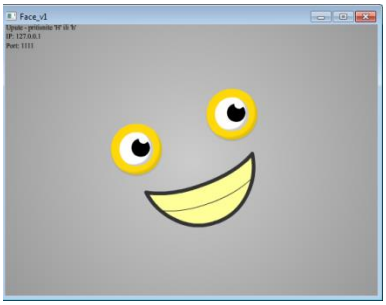
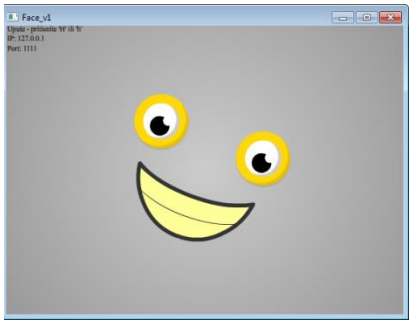


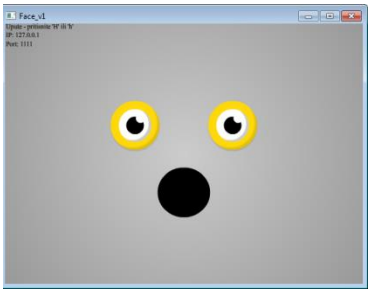
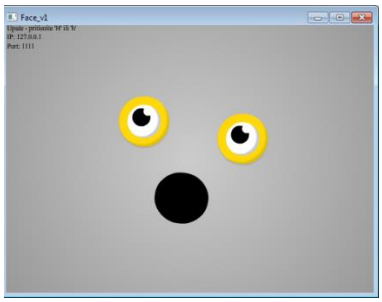
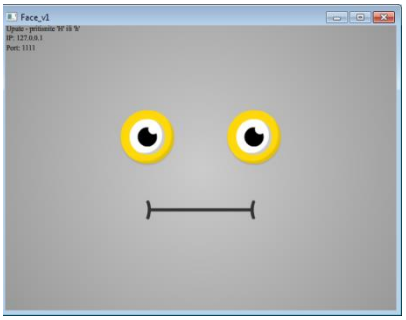
Slika 42. Primjer praktične funkcije - kada robot miče ruku desno, lice se usmjeri 'udesno' da to vizualizira.

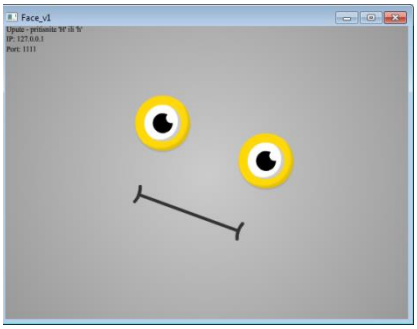
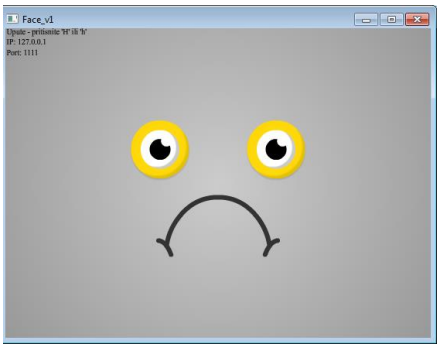
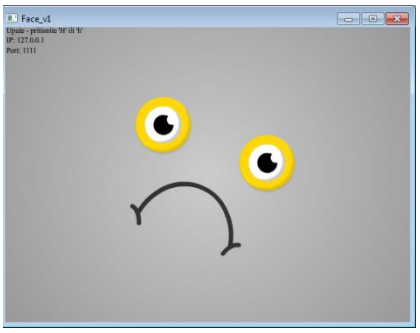
6.2. Popis vizualnih stanja

U sljedećoj tablici su prikazani primjeri neke od mogućih vizualnih stanja (tj. izraza lica) koje je program je u stanju prikazati, uz nekoliko pripadajućih ulaznih kodova (tj. informacija dobivenih od robota), te moguću interpretaciju (s obzirom na stanje samog robota) vizualnog stanja od strane korisnika.

Tablica 1. Popis vizualnih stanja, ulaznih kodova, te interpretacija.

VIZUALNO STANJE	ULAZNI KOD	INTERPRETACIJA
	100, 200, 259, 300	- Stanje pri početnom pokretanju programa ('pozdrav' za korisnika). - Stanje nakon završetka kretnji ili pokreta robota ('zadaća uspješno obavljena'). - Centralna ili početna koordinata robota, što se tiče pomičnih dijelova (npr. za robotsku ruku). - Potvrda o stabilnom ili pravilnom stanju.
	101, 201, 301	- Stanje nakon uspješnog kretanja udesno ('uspješno obavljen pomak udesno'). - Trenutna koordinatna pozicija nekog dijela robota (npr. robotska ruka) je 'udesno' od početne ili prethodne točke.
	111, 211, 311	- Kut nekog specifičnog dijela robota (npr. zglob robotske ruke) je lagano nagnut ulijevo, u odnosu na početnu ili prethodnu poziciju.
	197, 297, 397	- Kut okretnog dijela je nagnut udesno u odnosu na početno/prethodno stanje, pozicija pokretnog dijela je dolje/ispod/južno od početnog, te je radnja uspješno obavljena.

	400, 500, 559, 600	- Vizualno stanje koje se prikazuje u tijeku izvršavanja neke zadaće ili radnje koje traju neko vrijeme (npr. kretanje robotske ruke). - 'Radnja u tijeku'. - Kretnja u tijeku, gdje je konačna točka u koordinatnom središtu robota. - Kretnja u tijeku, gdje je trenutna (tj. privremena, uslijed kretanja) pozicija u središtu.
	475, 574, 674	- Informacija da robot trenutno obavlja kretnju u smjeru sjeveroistok, te da je okretni dio pod blagim kutom udesno.
	0, 00, 000	- Informacija da robot očekuje naredbe, odgovor na neki upit ili dodatnu informaciju od korisnika. - Oznaka da je robot namjerno prekinuo mrežnu vezu, ili da robot nije primio naredbe neko specifično vrijeme. - Stanje kada robot obavlja neku računsku ili matematičku operaciju, ili računalni proces, koji traje dulje od sekunde, te ne uključuje kretanje.

	99, 099	- Stanje koje prikazuje robot ukoliko se specifično upravlja okretnim dijelom robota (npr. zglobom), bez pomaka ostalih dijelova. - Stanje za prikaz pri ručnom namještavanju robota u neku poziciju.
	900	- Upozorenje o nekoj traženoj radnji. - Upozorenje da je tražena radnja nemoguća, ili da može dovesti do oštećenja robota ili opasnosti za korisnika. - Upozorenje da je došlo do greške pri odvijanju neke radnje, te da se morala odmah prekinuti zbog sigurnosti.
	999, bilo koji nevažeći kod	- Upozorenje o neprikladnoj ulaznoj informaciji, tj. obavijest da program je primio nevažeći ili neprepoznatljivu informaciju od robota s kojim je spojen. - Obavijest da je robot oštećen ili u drugom stanju pri kojemu nije u mogućnosti obaviti neku radnju. - Informacija da je neočekivano došlo do prekida mrežne veze s robotom.

7. ZAKLJUČAK

Razvijena programska podrška za humanoidnu vizualizaciju radnih stanja robota ispunjava postavljene zahtjeve zadatka. Razvijeni program u stanju je spojiti se s robotom preko mrežne veze, prikazuje približno tisuću mogućih vizualnih stanja kao reakciju na ulazne informacije od robota, te je kao program stabilan, funkcionalan, lako razumljiv i jednostavan za korištenje.

Nadalje je moguće isti program naknadno programirati da bi mu se promijenile značajke mrežne veze, radnog okruženja, ili prikazanih vizualnih stanja. Time se može osigurati kompatibilnost s raznim vrstama robota, te osigurati funkcionalnost programa za različite vrste računala.

Za svrhe programiranja programa, izrade likova i animacija, te zahtjeva mrežne veze, korišteni su programski alati za Adobe Flash i Adobe AIR razvojno okruženje, od tvrtke Adobe Systems. Za samo programiranje se koristio alat za izradu Adobe Flash programa, tzv. Flash Professional, te Actionscript 3 programski jezik.

8. LITERATURA

[1] Http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0072589

[2] Http://www.rethinkrobotics.com/baxter/

[3] Http://en.wikipedia.org/wiki/Uncanny_valley

[4] Http://www.adobe.com/products/flash.html

[5] Http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/

[6] Http://www.drk.com.ar/builder.php

9. PRILOZI

I. CD-R disketa s programom i ovim dokumentom u .pdf obliku

II. Ukupni programski kod:

```
import flash.text.TextField;

import flash.text.TextFieldAutoSize;

import flash.events.MouseEvent;

import fl.transitions.Tween;

import fl.transitions.*;

import fl.transitions.easing.*;


var beginrot

var endrot

var begin_x

var end_x

var begin_y

var end_y

var mouthexpression:int;

mouthexpression = 1;


var tf:TextField = new TextField();

tf.wordWrap = true;

tf.width = 300;

tf.height = 180;

tf.autoSize = TextFieldAutoSize.NONE;

logbox.addChild(tf);

logbox.visible = !logbox.visible;
```

```
function log(str) { tf.appendText("\n" + str); }

stage.addEventListener(KeyboardEvent.KEY_DOWN,logboxhandler);

function logboxhandler(event:KeyboardEvent){

        if(event.keyCode == 76 || event.charCode == Keyboard.L){

        // Sets the visibility of it to the opposite of what it is (starts off).

        logbox.visible = !logbox.visible;

        // Enable mouse events/selection based on visibility (it's all just booleans)

        logbox.enable = logbox.visible;         }}


helpscreen.visible = !helpscreen.visible;

stage.addEventListener(KeyboardEvent.KEY_DOWN,helpboxhandler);

function helpboxhandler(event:KeyboardEvent){

        if(event.keyCode == 72 || event.keyCode == Keyboard.H){

         // Sets the visibility of it to the opposite of what it is (starts off).

        helpscreen.visible = !helpscreen.visible;

         // Enable mouse events/selection based on visibility (it's all just booleans)

        helpscreen.enable = helpscreen.visible;         }}


import flash.events.SecurityErrorEvent;

import flash.system.Security;

import flash.display.Sprite;

import flash.events.Event;

import flash.events.IOErrorEvent;

import flash.events.ProgressEvent;

import flash.events.ServerSocketConnectEvent;

import flash.net.ServerSocket;

import flash.net.Socket;

import flash.utils.Timer;
```

```
import flash.events.TimerEvent;


var serverSocket:ServerSocket;
var clientSockets:Array = new Array();
ServerSocketSmiley();
function ServerSocketSmiley()
{ try
        {serverSocket = new ServerSocket();
                serverSocket.addEventListener( Event.CONNECT, connectHandler );
                serverSocket.addEventListener( Event.CLOSE, onClose );
                serverSocket.bind(1111, "127.0.0.1");
                trace ("Binding Socket.");
        log ("Binding Socket.");
        trace ("IP: " + serverSocket.localAddress);
        log ("IP: " + serverSocket.localAddress);
        trace ("Port: " + serverSocket.localPort);
        log ("Port: " + serverSocket.localPort);
                serverSocket.listen();
                log( "Listening on port " + serverSocket.localPort +"..."); }
        catch(e:SecurityError) { log(e); } }

function connectHandler(event:ServerSocketConnectEvent):void {var socket:Socket = event.socket as Socket;
        clientSockets.push( socket );
        socket.addEventListener( ProgressEvent.SOCKET_DATA, socketDataHandler);
        socket.addEventListener( Event.CLOSE, onClientClose );
        socket.addEventListener( IOErrorEvent.IO_ERROR, onIOError );
        socket.writeUTFBytes("| Connection established to server: "+serverSocket.localAddress +", port
"+serverSocket.localPort +". |");
```

```
            socket.flush();
            log("Sending connect message ('Connection established.')");

            var t:Timer = new Timer(300000, 1);
            t.addEventListener(TimerEvent.TIMER, handleTimer);
            t.start();}

function handleTimer(e:TimerEvent):void{     log("LOG: 5 minutes have elapsed on the server. Sending message to client.");
   clientSockets[0].writeUTFBytes("| TIMER MESSAGE: 5 minutes have elapsed since connection to port "+ serverSocket.localPort +". |");
   clientSockets[0].flush();}

function socketDataHandler(event:ProgressEvent):void { var socket:Socket = event.target as Socket;
            trace(socket.bytesAvailable + ' byte(s) available');
            log(socket.bytesAvailable + ' byte(s) available');

            if (socket.bytesAvailable > 0)
            {var msg:String = socket.readUTFBytes( socket.bytesAvailable );
            log( "Received: " + msg);

                        var sentnumber:Number = Number(msg);

                        trace("Is the sent value a NaN? Response: " + isNaN(sentnumber))
                        log("Is the sent value a NaN? Response: " + isNaN(sentnumber))

                        if (isNaN(sentnumber) == true)
```

```
{       sentnumber = 0
        trace ("Sent value was NaN, converting to zero.");
        log ("Sent value was NaN, converting to zero.");
        trace ("Converted to number: " + sentnumber);
        log ("Converted to number: " + sentnumber);}

else if (sentnumber > 999)
{       sentnumber = 999
        trace ("Sent value was too high, converting to maximum allowed.");
        log ("Sent value was too high, converting to maximum allowed.");
        trace ("Converted to number: " + sentnumber);
        log ("Converted to number: " + sentnumber);}

        else if (sentnumber < 0)
{       sentnumber = 0
        trace ("Sent value was negative, converting to zero.");
        log ("Sent value was negative, converting to zero.");
        trace ("Converted to number: " + sentnumber);
        log ("Converted to number: " + sentnumber);}
// Otherwise, the 0-999 number is used to alter the smiley and digit boxes.
else {  trace("Converted to number: " + sentnumber);
        log("Converted to number: " + sentnumber); }

socket.writeUTFBytes("| Echo: " + msg + " |");

socket.flush();
log( "Sending: " + msg );}
```

```
            var code:Number = Number(sentnumber);
            var jedinica:Number = Math.floor(code) % 10;
            var desetka:Number = Math.floor(code/10^0) % 10;
            var stotinjka:Number = Math.floor(code/100);

 if (stotinjka == 0) {
                        if (mouthexpression == 1)
                {       wholebody.mouth.gotoAndStop("Revsmile");        }
                if (mouthexpression == 4)
                {       wholebody.mouth.gotoAndStop("Revgasp");         }
                if (mouthexpression == 2)
                {       wholebody.mouth.gotoAndStop("Revnope");         }
// Tells it to respond by changing expression and making text.

            var myTimer1:Timer = new Timer(500,1);
            myTimer1.addEventListener(TimerEvent.TIMER, timerListener1);
            function timerListener1 (e:TimerEvent):void
            {
            trace("Timer is Triggered"); }
            myTimer1.start();
            myTimer1.addEventListener(TimerEvent.TIMER_COMPLETE, timerDone1);
            function timerDone1(e:TimerEvent):void
            {       trace("Timer finishing!");
                        mouthexpression = 3;
                        trace(mouthexpression);
                wholebody.mouth.gotoAndStop("Neutral");         } }

// Smile = 1, Nope = 2, Neutral = 3, Gasp = 4
```

```
else if (stotinjka >= 1 && stotinjka <=3) {
        if (mouthexpression == 3)
        {       wholebody.mouth.gotoAndStop("Revneutral");    }
        if (mouthexpression == 4)
        {       wholebody.mouth.gotoAndStop("Revgasp");    }
        if (mouthexpression == 2)
        {       wholebody.mouth.gotoAndStop("Revnope");    }
          var myTimer2:Timer = new Timer(500,1);
myTimer2.addEventListener(TimerEvent.TIMER, timerListener2);
function timerListener2 (e:TimerEvent):void
{trace("Timer is Triggered");            }
myTimer2.start();
myTimer2.addEventListener(TimerEvent.TIMER_COMPLETE, timerDone2);
function timerDone2(e:TimerEvent):void
{       trace("Timer finishing!");
        mouthexpression = 1;
        trace(mouthexpression);
        wholebody.mouth.gotoAndStop("Smile");    }   }
// Smile = 1, Nope = 2, Neutral = 3, Gasp = 4
  else if (stotinjka > 3 && stotinjka <= 6){
        if (mouthexpression == 1)
        {       wholebody.mouth.gotoAndStop("Revsmile");            }
        if (mouthexpression == 3)
        {       wholebody.mouth.gotoAndStop("Revneutral");          }
        if (mouthexpression == 2)
        {       wholebody.mouth.gotoAndStop("Revnope");    }
  var myTimer3:Timer = new Timer(500,1);
myTimer3.addEventListener(TimerEvent.TIMER, timerListener3);
```

```
function timerListener3 (e:TimerEvent):void
{trace("Timer is Triggered");}
myTimer3.start();
myTimer3.addEventListener(TimerEvent.TIMER_COMPLETE, timerDone3);
function timerDone3(e:TimerEvent):void
{           trace("Timer finishing!");
            mouthexpression = 4;
            trace(mouthexpression);
            wholebody.mouth.gotoAndStop("Gasp");    } }
// Smile = 1, Nope = 2, Neutral = 3, Gasp = 4
  else if (stotinjka > 6 && stotinjka <= 9){
            if (mouthexpression == 1)
            {       wholebody.mouth.gotoAndStop("Revsmile");   }
            if (mouthexpression == 4)
            {       wholebody.mouth.gotoAndStop("Revgasp");            }
            if (mouthexpression == 3)
            {       wholebody.mouth.gotoAndStop("Revneutral");         }
            var myTimer4:Timer = new Timer(500,1);
myTimer4.addEventListener(TimerEvent.TIMER, timerListener4);
function timerListener4 (e:TimerEvent):void
{trace("Timer is Triggered");           }
myTimer4.start();
myTimer4.addEventListener(TimerEvent.TIMER_COMPLETE, timerDone4);
function timerDone4(e:TimerEvent):void
{trace("Timer finishing!");
            mouthexpression = 2;
            trace(mouthexpression);
            wholebody.mouth.gotoAndStop("NOPE");  }   }
```

```
// Checks the 10's numbers, and rotates the smiley for each.


// Again, it first checks the zero...
if (desetka == 0) {endrot = 0
        var rot0:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1, true);
        beginrot = endrot;  }
         else if (desetka == 1) {  endrot = -20
                var rot10:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
                beginrot = endrot;          }
         else if (desetka == 2) {  endrot = -15
                var rot20:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
               beginrot = endrot;           }
         else if (desetka == 3) {  endrot = -10
                var rto30:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
               beginrot = endrot;           }
         else if (desetka == 4) {  endrot = -5
                var rot40:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
               beginrot = endrot;           }
         else if (desetka == 5) {  endrot = 0
                var rot50:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
               beginrot = endrot;           }
         else if (desetka == 6) {  endrot = 5
                var rot60:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
```

```
                beginrot = endrot;          }
        else if (desetka == 7) {  endrot = 10
                var rot70:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
                beginrot = endrot;          }
        else if (desetka == 8) {  endrot = 15
                var rot80:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
                beginrot = endrot;          }
        else if (desetka == 9) {  endrot = 20
                var rot90:Tween = new Tween(wholebody, "rotation", Strong.easeOut, beginrot, endrot, 1,
true);
                beginrot = endrot;          }

// Checks the 1's numbers, and moves the eyes of the smiley for each.
if (jedinica == 0) {end_x = 0
        end_y = 0
        var eye_x0:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
        var eye_y0:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
        var eye2_x0:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
        var eye2_y0:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
        begin_x = end_x
        begin_y = end_y  }
         else if (jedinica == 1) {
         end_x = 5
```

```
        end_y=0
        var eye_x1:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x,end_x,1,true);
        var eye_y1:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut,
begin_y,end_y,1,true);
        var eye2_x1:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x,end_x,1,true);
        var eye2_y1:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y,end_y,1,true);
        begin_x=end_x
        begin_y=end_y}
         else if (jedinica==2){
        end_x=4
        end_y=-4
        var eye_x2:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x,end_x,1,true);
        var eye_y2:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut,
begin_y,end_y,1,true);
        var eye2_x2:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x,end_x,1,true);
        var eye2_y2:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y,end_y,1,true);
        begin_x=end_x
        begin_y=end_y}
         else if (jedinica==3){
        end_x=0
        end_y=-5
        var eye_x3:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x,end_x,1,true);
```

```
var eye_y3:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
var eye2_x3:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
var eye2_y3:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
begin_x=end_x
begin_y=end_y}
 else if (jedinica==4){
end_x=-4
end_y=-4
var eye_x4:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
var eye_y4:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
var eye2_x4:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
var eye2_y4:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
begin_x=end_x
begin_y=end_y}
 else if (jedinica==5){
end_x=-5
end_y=0
var eye_x5:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut, begin_x, end_x, 1, true);
var eye_y5:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut, begin_y, end_y, 1, true);
```

```
var eye2_x5:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye2_y5:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
begin_x=end_x
begin_y=end_y}
 else if (jedinica==6) {
end_x=-4
end_y=4
var eye_x6:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye_y6:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
var eye2_x6:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye2_y6:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
begin_x=end_x
begin_y=end_y}
 else if (jedinica==7) {
end_x=0
end_y=5
var eye_x7:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye_y7:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
var eye2_x7:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
```

```
var eye2_y7:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
begin_x=end_x
begin_y=end_y}
else if (jedinica==8) {
end_x=4
end_y=4
var eye_x8:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye_y8:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
var eye2_x8:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye2_y8:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
begin_x=end_x
begin_y=end_y}
else if (jedinica==9) {end_x=0
end_y=0
var eye_x9:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye_y9:Tween = new Tween(wholebody.botheyes.eyesocket.eyeball.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
var eye2_x9:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "x", Strong.easeOut,
begin_x, end_x, 1, true);
var eye2_y9:Tween = new Tween(wholebody.botheyes.eyesocket2.eyeball2.pupil, "y", Strong.easeOut,
begin_y, end_y, 1, true);
begin_x=end_x
begin_y=end_y}}
```

```
function onClientClose( event:Event ):void { log("Connection to client closed."); }

function onIOError( errorEvent:IOErrorEvent ):void {log("IOError: " + errorEvent.text); }

function onClose( event:Event ):void { log("Server socket closed by OS."); }
```