Kodiranje u ActionScript jeziku

**Predmet: Računarska grafika**

# SADRŽAJ

1. UVOD U ACTIONSCRIPT 3

1.1 ISTORIJAT I KOMPATIBILNOST VERZIJA 4

1.2 DODAVANJE ACTIONSCRIPTA U FILMU 6

2. OSNOVNI POJMOVI 8

2.1 IME INSTANCE (Instance Name) 8

2.2 PROMENLJIVA (Variable) 8

2.3 DOMEN (Scope) 8

2.4 SINTAKSA TAČKE – putanja do objekta 9

2.5 PARAMETRI (Parameters) 10

2.6 NIZOVI (Arrays) 10

2.7 VITIČASTE ZAGRADE 11

2.8 FUNKCIJA (Function) 11

2.9 KLASA (Class) 12

2.10 METODI (Methods) 12

2.11 SVOJSTVA (Properties) 12

2. 12 SINTAKSA (Syntax) 13

3. ACTIONSCRIPT PROMENLJIVE I OPERATORI 14

3.1 ACTIONSCRIPT PROMENLJIVE (Variables) 14

3.2 DEKLARACIJA PROMENLJIVE 14

3.3 PRAVILA ZA IMENOVANJE PROMENLJIVIH 14

3.4 VRSTE PROMENLJIVIH I PRIDRUŽIVANJE VREDNOSTI 15

3.5 MATEMATIČKI OPERATORI I PROMENLJIVE 16

3.6 OPERATORI 17

4. UPRAVLJANJE TOKOM FILMA 19

4.1 AKCIJE stop() i play(); 19

4.2 AKCIJE gotoAndPlay() i gotoAndStop() 20

4.3 AKCIJE nextFrame() i prevFrame() 21

4.4 AKCIJE nextScene() i prevScene() 21

5. USLOVNE AKCIJE 22

5.1 AKCIJA on(); 22

5.2 AKCIJA if...else 24

5.3 AKCIJA switch…case 27

5.4 AKCIJA for 28

5.5 DELOVI SKRIPTI 28

6. KOMUNIKACIJA SA SPOLJNIM FAJLOVIMA 30 6.1 AKCIJA getURL() 30

6.2 AKCIJA loadVariables 31

6.3 AKCIJA loadMovie(); 33

7. KORISNIČKE FUNKCIJE i onClipEvent() 34

7.1 FUNKCIJE 34

7.2 onClipEvent() 36

8. REUZIMANJE PODATAKA OD KORISNIKA 39

8.1 OTKRIVANJE PRITISKA NA TASTERE 39

8.2 DETEKCIJA PRITISKA POMOĆU TASTERA 39

8.3 Key OBJEKT 40

8.4 Key Listeners 40

8.5 UPIS I ISPIS TEKSTA (Input Text) i (Dynamic Text) 41

8.6 STRING OPERACIJE 44

8.7 FUNKCIJE STRINGA 44

9. OBJEKTNO ORJENTISANO PROGRAMIRANJE 46

9.1 OBJEKTI, SVOJSTA I METODI 46

9.2 KREIRANJE I UPOTREBA OBJEKATA 47

9.3 SPISAK UGRAĐENIH OBJEKATA SA SVOJSTVIMA I METODIMA 49

10. PRIMERI U ACTIONSCRIPTU 52

# 

# 1. UVOD U ACTIONSCRIPT

**Actionscript je programski jezik** za Adobe Flash i Adobe AIR Runtime okruženje. On omogućava interaktivnost, obradu podataka kao i mnogo drugih opcija u Flashu, Flexu i AIR aplikacijama.

Actionscript se izvršava pomoću Actionscript Virtual Machine (AVM) koja je deo Flash Playera. Kod se kompajlira, prebacuje se u bytecode format koji se ugrađuje u izlazni SWF fajl, a kasnije izvršava od strane Flash Playera.

Programski jezik je lista instrukcija koja govori računaru sta treba da uradi. Računar izvršava ove instrukcije tačno onako kako su navedene. Lista instrukcija za računar se zove program. Drugi naziv za program je skripta (kod). Program ili skripta postoje unutar nekog okruženja. U slučaju ActionScripta to okruženje je Flash film. Skripta može reći Flash filmu šta da učini. U nekim slučajevima, skripte mogu zahtevati izvršenje i nekih drugih operacija, kao recimo, šta korisnički Web pretraživač ili operativni sistem treba da urade.

Skripte mogu biti veličine jednog reda, a mogu biti duge i na stotine redova. Mogu bii smeštene u jednom centralnom delu Flash filma ili razbacane po njemu. Neki ljudi će koristiti ceo Flash film koji koriste skripte kao program, dok će ostali kao program koristiti individualne skripte u filmu. Obe definicije su tehnički ispravne iz razloga što jedan program može biti određen sa više manjih programa. Najvećim delom, skripte su mali programi smešteni u različitim frejmovima i elementima unutar Flash filma.

Flash film se sastoji od scena. Svaka scena ima svoju vremensku zonu. Svaka vremenska zona počinje sa kadrom1 i nastavlja se odatle. Prirodno stanje Flash filma je da se kreće unapred konstantnom brzinom od scene1, kadra1 pa do kraja scene1 i zatim prelazi na scenu2 i tako dalje. Može se podesiti da se film kreće kroz sve kadrove svih scena i zatim ga zaustaviti. Ili, da se neprekidno vrti. Osnovna svrha ActionScripta je da promeni ovo linearno ponašanje.

Neke od osnovnih funkcija koje ActionScript može izvršavati su:

* **Animacija –** ActionScript nije potreban za kreiranje animacija u Flashu. Ali skriptovanje može pomoći pri kreiranju nekih kompleksnijih animacija. Na primer, pomoću ActionScripta lopta se može kretati po ekranu u naizgled beskrajnoj putanji. Ukoliko pravimo animacije ali bez pomoći ActionScripta, potrebno nam je na desetine frejma.Ali skripte sve to mogu učiniti samo jednom frejmu.
* **Navigacija –** Umesto kreiranja filma konstantnom brzinom, film se može pauzirati i može se korisniku prepustiti odluka odakle će ga nanovo pustiti.
* **Korisnički ulaz –** Korisniku se mogu postavljati pitanja i zatim se te informacije mogu koristiti u filmu ili ih poslati na server. Flash film u kombinaciji sa ActionScriptom je bolji način za popunjavanje Web formi. Ili, film sam po sebi može koristiti određene informacije da bi se prilagodio korisnikovom iskustvu.
* **Preuzimanje podataka –** Skripta može unakrsno komunicirati sa serverom. Tako možemo preuzeti ažurirane podatke i prezentirati ih korisniku.
* **Izračunavanja –** ActionScript može obavljati i neka izračunavanja. Recimo, izračunavanje rate kredita ili dodavanje troškova nabavljenih stvari na kreditnu karticu.
* **Grafičke izmene –** Skripte se mogu napisati tako da se njima mogu menjati veličina, ugao rotacije ili čak boja filmskih klipova. Takođe se mogu i uklanjati delovi scena i mnogo drugih stvari.
* **Istraživanje okruženja –** AcionScript se može koristiti za istraživanje okruženja u kom se film pušta.
* **Zvuk –** ActionScript je dobra alternative Flashovoj vremenskoj zoni za puštanje zvuka. Čak se mogu kontrolisati i balans i jačina zvuka.

## 1.1 ISTORIJAT I KOMPATIBILNOST VERZIJA

Prva verzija Actionscripta (verzija 1.0) je dodata Flashu kako bi se omogućila interakcija korisnika sa Flash filmom i kako bi se omogućila kontrola puštanja određenih delova filma. Od tada, Actionscript se razvijao zajedno sa Flashom kao alatom i time se dobio alat koji poseduje snažan programski jezik koji proširuje mogućnosti Flasha.

Verzija 1.0 Actionscripta je bila *sekvencijalni* programski jezik, što znači da su instrukcije dodavane liniju po liniju bez mogućnosti višestruke upotrebe. Bilo je moguće uneti kod i dodati ga simbolu ili frejmu, ali je bilo neophodno dodavati kod zasebno svakom elementu koji želimo da učestvuje u interakciji.

Nakon početnog uspeha, lansirana je verzija 2.0 koja se oslanjala na OOP (objektno orjentisano programiranje). No, govoriti o Actionscriptu 2.0 kao OOP jeziku je bilo diskutabilno. Ta verzija zaista jeste bila objektno orjentisana, ali na početnom nivou koji je jedva razdvajao sekvencijalno od objektno orjentisanog programiranja. Ono što je Actionscript 2.0 svakako uneo jesu procedure (zato se za Actionscript 2.0 kaže da je *proceduralni* jezik). Naime, tako je bilo moguće dodavati funkcije koje će se uneti samo na jednom mestu a koristiti na više mesta u filmu. No, svakako da je doneo prekretnicu, a s obzirom da se mogao kombinovati sa Actionscript 1.0 verzijom, stekao je masovnu popularnost.

U potrazi za još većom modularnošću koda i još većim stepenom ponovne upotrebe, Actionscript je gravitirao ka OOP jeziku. Objekti su individualne instance klasa objekata (*Class*) – kolekcija koda koja je nezavisna u odnosu na druge objekte i koja ne utiče na njih. Ograničavanje koda na male *kapsule* je donelo koncept enkapsulacije (*Encapsulation*a) koji je jedan od glavnih obeležja objektno orjentisanog programiranja. Drugi značajan koncept je nasleđivanje (*Inheritance*), mogućnost izvođenja klasa iz „viših“ (*Parent -* roditelj) klasa, tj. nasleđivanje specifičnih karakteristika „klase roditelja“.

OOP je svakako najbolji koncept programiranja za veoma velike projekte, kao i za istovremeni rad više programera, ali može biti nezgodan za male projekte iz prostog razloga što veliki broj mogućnosti opterećuje jednostavne zadatke. Takođe, proces učenja Actionscript 3.0 jezika je značajno duži.

Ukratko, OOP nije uvek najbolji izbor za određeni posao. Proceduralno programiranje i dalje ima svoju ulogu, a Adobe Flash od verzije CS3 nudi mogućnost da se upoznate i iskoristite oba principa programiranja.

U verziji CS3, Adobe je integrisao najnoviju verziju Actionscript programskog jezika koji nosi oznaku 3.0. Reč je o kompletno objektno orjentisanom programskom jeziku koji ima potpuno drugačiju sintaksu i komande i koji drastično proširuje mogućnosti Flasha. Naime, uz pojavu Flexa, akcenat je stavljen na programiranje pa je Actionscript 3.0 došao u pravom trenutku.

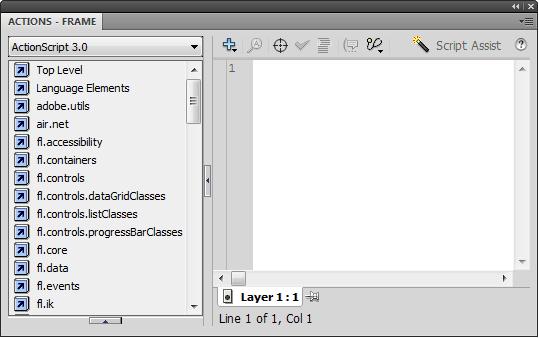
Kao što je rečeno, Actionscript 3.0 donosi u potpunosti drugačiju sintaksu i koncept programiranja i iz tog razloga ga je nemoguće kombinovati sa prethodnim verzijama. Za posledicu, pri kreiranju novog Flash dokumenta morate izabrati za koju verziju Actionscript jezika kreirate film.

Što se tiče svih problema u kompatibilnosti između verzije 3.0 i ranijih verzija Actionscripta, najbitniji su:

* SWF fajl ne može kombinovati Actionscript 1.0 ili 2.0 kod sa Actionscript 3.0 kodom,
* Actionscript 3.0 kod može učitati fajl pisan Actionscript 1.0 ili 2.0 kodom, ali ne može pristupiti njegovim varijablama i funkcijama,
* SWF fajlovi pisani Actionscript 1.0 ili 2.0 verzijama ne mogu učitati fajlove pisane u Actionscript 3.0 jeziku. To znači da svi SWF fajlovi kreirani u Flash 8, odnosno Flex Builderu 1.5 ili ranijim ne mogu učitati fajlove kreirane u verzijama CS3 i CS4 koji koriste Actionscript 3.0,
* u principu, SWF fajlovi pisani u Actionscriptu 1.0 ili 2.0 moraju biti prebačeni u Actionscript 3.0 kako bi radili sa fajlovima pisanim u Actionscriptu 3.0. Recimo, ako kreirate Media player pisan u Actionscript 2.0 jeziku, koji učitava razne sadržaje putem Actionscript 2.0. Ne možete kreirati novi sadržaje Actionscriptom 3.0 i učitati ih u plejer, već najpre morate Actionscript 2.0 kod prepraviti u Actionscript 3.0 kod.

## 1.2 DODAVANJE ACTIONSCRIPTA U FILMU

Bilo da je reč o Actionscript 1.0 i 2.0 ili o verziji Actionscript 3.0, za dodavanje akcija nam služi Actions panel koji pozivamo iz menija *Window > Action* ili prečicom F9.



*slika 1.1 – ActionScript panel*

U verziji 1.0 Actionscript se dodavao direktno simbolima na sceni, dok se kod verzija 2.0, a posebno kod verzije 3.0 preporučuje kreiranje novog sloja (uobičajeno se zove *Actions*), a zatim kucanje koda u prvom frejmu tog sloja. Na taj način se lako pronalazi kod i vrše korekcije koda. Ukoliko bi vam kod bio razbacan svuda po Timelineu, dosta vremena bi se gubilo na traženje mesta gde se nalazi problematični kod.

Još jedna bitan panel u Flashu je i *Output* panel. On se otvara automatski pored Timeline panela u trenutku kada se u njemu pojavi neki ispis. Output panel ispisuje poruke o problemima u našem filmu u trenutku kada testiramo film. Takođe, ovde je moguće ispisati i vrednosti nekih promenljivih, kao i običan tekst.

# 2. OSNOVNI POJMOVI

**2.1 IME INSTANCE (Instance Name)**

Svaki simbol se može pojaviti više puta na sceni (u istom ili različitim trenucima filma). Da bi se Actionscript-om kontrolisali parametri vezani za simbol, svakom simbolu se mora dodeliti ime tog pojavljivanja (instance) simbola. Instance Name menjate selekcijom simbola, a zatim upisom imena u odgovarajućem polju u Properties panelu.

**2.2 PROMENLJIVA (Variable)**

Promenljiva predstavlja tip podatka koji najčešće nije konstantan. Kada definišete promenljivu, ujedno definišete i tip podatka, što odreñuje koje podatke će promenljiva predstavljati. Promenljiva String može sadržati i brojeve i slova, dok promenljiva Number može sadržati samo brojeve.

*var promenljiva=1;*

Ako dodeljujete String nekoj varijabli, morate koristiti navodnike:

*var ime="Petar";*

Ne morate definisati tip promenljive jer će Flash automatski prepoznati da li je reč o broju ili nizu slova. Promenljive moraju imati jedinstveno ime i morate voditi računa o malim i velikim slovima. Promenljive mogu sadržati samo brojeve, slova i donje crte, a ne mogu počinjati brojem. Imena promenljivih ne mogu da sadrže razmake.

**2.3 DOMEN (Scope)**

Domen definiše oblast u Flash fajlu u kome se promenljive mogu referencirati. Promenljive mogu biti lokalne, globalne ili Timeline (vremenske).

* **Lokalne promenljive** ukazuju samo na jednu oblast, na primer frejm ili funkciju.

Kada prestane izvršavanje te oblasti, lokalna promenljiva nestaje. Unutar funkcije,

lokalne promenljive se definišu unutar vitičastih zagrada {}.

* **Globalne promenljive** su dostupne bilo gde, bilo kada i mogu ukazivati na bilo šta.

Možemo ih definisati, a zatim koristiti na ostalim frejmovima pa čak i drugim SWF fajlovima koji su smešteni u glavni Movie fajl. One se definisu upotrebom prefiksa \_global.

* **Timeline promenljive** su dostupne bilo kom skriptu u okviru istog Timeline-a. Pošto je promenljiva definisana na Timeline-u, dostupna je svim frejmovima koji se u njoj nalaze.

**2.4 SINTAKSA TAČKE – putanja do objekta**

Actionscript može da kontroliše određene parametre simbola. Međutim, tu nam nisu od koristi imena simbola, s obzirom na to da se jedan simbol može više puta pojaviti na sceni. Zato je uveden koncept *Instance Name* koji svakom simbolu dodeljuje jedinstveno ime. Kako bi Actionscript-om mogli da kontrolišete neki simbol, morate znati kako da postavite putanju do tog simbola.

Glavna scena svakog filma je *\_root* nivo. Ovo je osnovni nivo i možete postaviti putanju tako da uvek kreće od ovog osnovnog nivoa. Ako na glavnoj sceni imate neki simbol čiji je Instance Name, na primer, *Krug* (dakle, Instance Name, a ne ime simbola), putanja do tog simbola je:

\_root.Krug

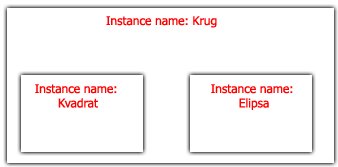
Primetićete tačku između ove dve reči. Tačka mora da stoji tu i nisu dozvoljeni razmaci jer ceo izraz čini putanju. Sada pretpostavite da u okviru simbola koji se nalazi na glavnoj sceni i ima *Instance Name Krug* imamo još jedan simbol koji ima *Instance Name Kvadrat*. Dakle, u simbolu *Krug* imate simbol *Kvadrat* (i dalje pričamo o *Instance Name-u*). Putanja do takvog simbola bi bila:

\_root.Krug.Kvadrat

Ako bih poželeo da promenim X koordinatu tog simbola, akcija bi izgledala ovako:

\_root.Krug.Kvadrat.\_x = 100;

Parametar \_x je rezervisana reč koja se odnosi na X koordinatu simbola.



Dakle, ako iz Timeline-a simbola *Kvadrat* želimo da editujemo X koordinatu simbola *Elipsa*, to možemo učiniti na dva načina:

apsolutnom adresom: \_root.Krug.Elipsa.\_x

ili relativnom adresom: \_parent.Elipsa.\_x

Relativnom adresom *\_parent* „izlazimo“ jedan nivo iznad, a zatim u tom nivou tražimo simbol određenog Instance Name-a. Takođe, možemo više puta upotrebiti *\_parent*:

\_parent.\_parent.\_parent.imesimbola

Ovime ćemo izaći tri nivoa iznad nivoa na kom dodajemo kod, a zatim ćemo tu kontrolisati simbol. Ukoliko se nalazimo na Timeline-u simbola Krug i želimo da kontrolišemo X koordinatu simbola Elipsa, biće dovoljno samo da upišemo:

elipsa.\_x = 100;

**2.5 PARAMETRI (Parameters)**

Parametri omogućuju dodavanje specifičnih detalja zadavanjem vrednosti između zagrada, u liniji koda. Na primer, u kodu gotoAndPlay(5); parametar 5 upućuje skript da pređe na frejm 5.

**2.6 NIZOVI (Arrays)**

Nizovi služe za čuvanje liste informacija. Određenom elementu u nizu pristupate pozivajući se na njegov indeks. Ime niza može biti recimo *sportovi*, a pet vrsta aktivnosti bili bi elementi niza. Indeks ukazuje gde se element nalazi u redu.

Kreiranje niza je lako:

sportovi = new Array();

ili

voce = new Array("jabuke","kruske","banane");

Tako kreiramo novi niz sa tri elementa:

voce[0] == jabuke;

voce[1] == kruske;

voce[2] == banane;

Primećujemo da indeksi počinju nulom.

**2.7 VITIČASTE ZAGRADE**

Vitičaste zagrade se koriste za grupisanje akcija u okviru neke funkcije. Ako recimo imate funkciju za sabiranje dva broja:

function saberi(a,b) {

c=a+b;

return c;

}

Sve što se nalazi između otvorene i zatvorene vitičaste zagrade će se izvršiti po pozivanju funkcije. Moguće je imati „ugnježdene“ zagrade, a jedino pravilo koje treba ispoštovati je da svaka otvorena zagrada mora biti zatvorena u okviru skripta.

**2.8 FUNKCIJA (Function)**

Funkcija predstavlja grupu iskaza kojima pristupamo pomoću imena. Funkcija omogućava da koristite skup iskaza više puta, a da ne morate da kucate sve iskaze, već samo naziv funkcije. Na primer, ako imate skript koji ima jednu liniju koda (jedan iskaz), nije problem da ga otkucate ponovo za svaki taster ili neki drugi simbol. Ako imate skript koji ima mnogo više iskaza i želimo da primenimo taj skript za nekoliko tastera, mnogo je lakše da kreirate funkciju.

**2.9 KLASA (Class)**

Klasa predstavlja apstraktni prikaz objekta. Svaki objekat u ActionScript-u 3.0 je definisan klasom. Klasa je kategorija objekata koji imaju iste metode i svojstva. Ako na primer, imate klasu automobil, njegova svojstva da ima četiri točka i sl.

ActionScript sadrži preko 100 ugrađenih klasa koje su predefinisani tipovi podataka. Da biste pristupili metodima i svojstvima klase, potrebno je kreirati primerak (instancu) klase deklarisanjem promenljive i određivanjem tipa podataka.

**2.10 METODI (Methods)**

Metodi su ključne reči koji daju rezultat akcije. Na primer dva metoda koja su povezana sa MovieClip klasom su *stop()* i *gotoAndPlay()*.

**2.11 SVOJSTVA (Properties)**

Svojstva opisuju objekat u klasi. Na primer Marko, Ana i Nenad imaju određenu visinu, težinu i npr. boju kose.

Marko.\_visina=185;

Ana.\_tezina=55;

Nenad.\_boja kose=“crna”

Isto tako je i sa postojećim objektima u ActionScript-u:

* MovieClip.\_height – visina Movie Clip simbola
* MovieClip.\_x – pozicija Movie Clip simbola na sceni
* MovieClip.\_ rotation – rotacija Movie Clip simbola

Za menjanje lokacije i izgleda klipa može se koristit nekoliko atributa. Atributi \_x i \_y omogućavaju podešavanje horizontalne i vertikalne lokacije klipa. Za menjanje horizontalne skale klipa mogu se koristiti \_xscale i \_width i \_yscale i \_height za menjanje vertikalne skale klipa.

Rotacija klipa vrši se pomoću atributa \_rotation. Osim toga, atributima \_alpha i \_visible moguće je kreirati poluprovodan ili potpuno nevidljiv klip. Horizontalna i verikalna pozicija kursora se može odrediti atributima \_xmouse i \_ymouse.

**2. 12 SINTAKSA (Syntax)**

* Tačka i zarez (;) na kraju reda obaveštava ActionScript da je stigao do kraja linije

koda i da pređe u novu.

* Svaka otvorena zagrada mora da ima i odgovarajuću zatvorenu zagradu. Isto pravilo važi za sve tipove zagrada (), [], {}.
* Tačka (.) omogućava pristup svojstvima i metodima objekta. Otkucajte ime primerka (instance), zatim stavite tačku, zatim naziv svojstva ili metoda.
* Kada unosite niz karaktera (String) ili naziv fajla, koristite znake navoda.
* Kao i u drugim jezicima i u ActionScript-u možete dodavati komentare, koji služe kao podsetnik ili objašnjenje o tome šta ste kojim delom skripta želeli da postignete. Da biste dodali jednoredni komentar, započnite ga sa dve kose crte (//), a višeredni komentar započnite sa /\* i završite sa \*/.

# 3. ACTIONSCRIPT PROMENLJIVE I OPERATORI

**3.1 ACTIONSCRIPT PROMENLJIVE (Variables)**

U ActionScript-u postoje tri osnovna tipa promenljivih:

* Tekstualne (*String*)
* Numeričke (*Number*)
* Logičke (*Boolean*)

**3.2 DEKLARACIJA PROMENLJIVE**

Promenljiva se u ActionScript-u deklariše kombinacijom ključne reči *var*, zatim nazivom promenljive i tipom promenljive. Naziv i tip se odvajaju sa dve tačke (to je gramatičko pravilo ActionScript-a).

Upotreba ključne reči *var* i tipa promenljive prilikom deklaracije nije obavezna, ali je preporučljiva da ne bi dolazilo do grešaka u programiranju. Takođe, ako prilikom deklarisanja promenljive, navedemo njen tip, Actions Panel nam „pomaže“ pri pridruživanju metoda promenljivoj (nudi nam spisak iz kog možemo odabrati željenu metodu). Možete na primeru da vidite kako izgleda pravilno deklarisana promenljiva:

var broj:Number;

Znači, ovde smo primenili:

• Upotrebu ključne reči (var)

• Imenovanje promenljive (broj)

• Određivanje tipa promenljive (Number)

• Završetak naredbe (;)

**3.3 PRAVILA ZA IMENOVANJE PROMENLJIVIH**

U ActionScript-u postoje određena pravila za imenovanje promenljivih kojih se morate pridržavati, a to su:

* U nazivu promenljive možete koristiti samo mala i velika slova engleske abecede,

brojeve, donju crtu (\_) i znak $.

* Na prvom mestu u nazivu ne sme biti broj.
* Kao naziv promenljive ne možete koristiti ključne reči.
* Ne možete korisiti razmake.
* Morate paziti na upotrebu velikih i malih slova, jer je ActionScript Case Sensitive: var x:Number i var X:Number su različite promenljive.

**3.4 VRSTE PROMENLJIVIH I PRIDRUŽIVANJE VREDNOSTI**

U ActionScript-u imamo više vrsta promenljivih, a najčešće se se koriste numeričke (*Number*), zatim tekstualne (*String*) i logičke (*Boolean*). Vrednosti promenljivim, nakon što su deklarisane, pridružujemo upotrebom znaka jednakosti (=). Primer deklaracije i pridruživanja vrednosti promenljivoj:

var broj:Number;

broj=5

U ovom primeru možete videti:

* Deklaraciju promenljive ključnom reči (*var*)
* Imenovanje promenljive (*broj*)
* Određivanje tipa promenljive (*Number)*
* Pridruživanje vrednosti promenljivoj (*5*)

Ključnu reč *var* koristili ste samo kod deklaracije promenljive. Kad se jednom odredi vrsta promenljive, njoj se od tog trenutka može samo pridruživati vrednost tog tipa. Ako pokušate pridružiti promenljivoj vrednost nekog drugog tipa (tekst, logički), Flash će prijaviti grešku i naš program se neće kompajlirati. Da vidimo primer deklarisanja tekstualne promenljive:

var mojTekst:String;

mojTekst =”Visoka tehnicka skola”;

U ovom primeru deklarisali smo tekstualnu promenljivu (vrsta *String*), a kao vrednost smo joj pridružili tekst *”Visoka tehnica skola”*. Vrednosti se tekstualnim promenljivim uvek pridružuju pod navodnicima. Navodnici mogu biti jednostruki (’ ’) ili dvostruki (” ”), ali moramo biti dosledni pri njihovoj primjeni.

Primer deklarisanja logičke promenljive:

var logika:Boolean;

logika=true;

U ovom primeru deklarisali smo logičku promenljivu (vrsta *Boolean*), a kao vrednost smo joj pridružili *true* (istina). Logička varijabla može osim true imati još samo jednu vrednost, a to je *false* (laž).

**3.5 MATEMATIČKI OPERATORI I PROMENLJIVE**

U svakom programskom jeziku, pa tako i u ActionScript-u imamo matematičke operatore. U ActionScript-u oni su:

* operator sabiranja +
* operator oduzimanja -
* operator množenja \*
* operator deljenja /
* operator celobrojnog deljenja %

Do sada smo videli kako promenljivoj možemo pridružiti konstantu kao vrednost (broj=5),a nakon što smo se upoznali sa matematičkim operatorima u ActionScript-u možemo videti i drugi način pridruživanja vrednosti promenljivim. Sledi primer:

var x:Number;

var y:Number;

x=50;

y=x\*2 + 1;

Nakon što se izvrši ovaj kod, vrednost koja je sačuvana u promenljivoj y je 101.

U prethodnom primeru videli smo kako vrednosti promenljivima možemo pridružiti i pomoću matematičkih izraza. ActionScript izvršitelj prvo napravi proračun matematičkog izraza koji se nalazi na desnoj strani znaka jednakosti pa zatim rezultat te matematičke operacije smesti u promenljivu y. Sada ćemo videti primere sa operacijama nad promenljivama tekstualnog tipa:

var mojTekst1:String;

var mojTekst2:String;

var zbirTeksta:String;

mojTekst1="Adobe";

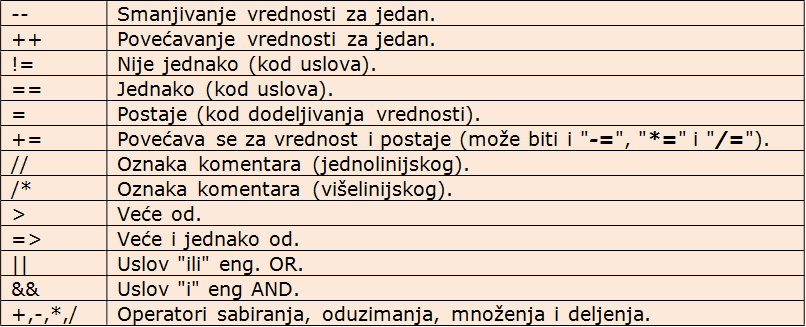
mojTekst2="Flash";

zbirTeksta=mojTekst1+mojTekst2;

Nakon što se izvrši gore navedeni kod, vrednost koja je sačuvana u promenljivoj zbirTeksta je "AdobeFlash".

**3.6 OPERATORI**

Prikaz najčešće korišćenih operatora u Actionscript-u:



Pogledajte prikazani deo koda i pokušajte da vidite možete li da odredite vrednost promenljive:

var a=7;

a++;

a-;

trace (a);

Rezultat je 7. 1 je dodato vrednosti *a,* a zatim je isto toliko oduzeto*.*

var a=7;

a+=6;

a =2;

trace(a);

Rezultat je 26. Početna vrednost promenljive *a*  je 7, a zatim promenjena na 13 kada joj je dodato 6, i zatim je ta nova vrednost pomnožena sa 2.

var a=22;

var b=6;

var c=13;

a = b+c;

a-=2;

b++;

trace(a);

Rezultat je 17. Početna vrednost *a* je 22, ali je zamenjena zbiroim *b* i *c.* Zatim je od *a* oduzeto 2 i ostalo je 17. *b* je promenjeno u 5 ali to nije uticalo na vrednost *a.*

# 4. UPRAVLJANJE TOKOM FILMA

Ono što je najjednostavnije, ali ujedno i najmoćnije u ActionScriptu je upravljanje tokom Flash filma. Moguće je zaustaviti film na određenom frejmu, ili prelaziti sa frejma na frejm. Ova akcija oslobađa Flash film od njegovog ravnomernog pravolinijskog kretanja koji mu je stalno određen.

Najvažnije akcije kojima možemo kontrolisati puštanje filma u okviru glavnog Timeline-a, kao i filma u okviru MovieClip simbola su :

* *stop();*
* *gotoAndPlay();*
* *gotoAndStop();*
* *play();*
* *nextFrame() i prevFrame();*
* *nextScene() i prevScene();*

**4.1 AKCIJE *stop()* i *play();***

Verovatno najjednostavnija akcija u ActionScriptu je akcija stop. Kada je izdata ona jednostavlja zaustavlja film u njegovom kretanju na onom frejmu (kadru) na kome se u tom trenutku nalazi.

U najvećem broju slučajeva, potrebno je da se akcija stop postavi u frejm unutar vremenske ose, a to možete učiniti ovom akcijom. Sintaksa je vrlo jednostavna:

stop();

Ovako upisana akcija u Actions panelu nekog ključnog frejma će zaustaviti glavu za čitanje na tom frejmu dokle god se ne aktivira neka druga akcija koja će ili pomeriti glavu za čitanje ili prebaciti glavu na neki drugi frejm.

Sve animacije u okviru klipova ili grafički simboli na frejmu nastavljaju animaciju, dok se animacija u okviru glavne vremenske zone zaustavlja. Za nastavljanje filma potrebno je izdavanje nove ActionScript akcije. Najednostavnija je akcija *play()*. Ona pokreće film, koji prelazi na sledeći frejm i zatim nastavlja kretanje.

Ukoliko želite da zaustavite animaciju unutar nekog *MovieClip* simbola, ne možete dodati akciju *stop()* na taj simbol, već morate „ući“ u simbol (editovati ga) i u Timeline-u tog simbola izabrati određeni ključni frejm i njemu dodeliti akciju *stop().*

Osim toga, ako imate neki *MovieClip* simbol na sceni, sa Instance Name-om *kvadrat* koji u sebi ima neku animaciju, sa glavnog Timeline-a možete zaustaviti animaciju u simbolu ako upotrebite Instance Name:

kvadrat.stop();

**4.2 AKCIJE *gotoAndPlay()* i *gotoAndStop()***

Još jedna od osnovnih naredbi ActionScripta je gotoAndPlay akcija. Pomoću ove naredbe prelazi se sa trenutnog aktivnog frejma na bilo koji drugi naznačeni frejm. Naznačivanje odredišnih frejmova se može vršiti po njihovom broju ili imenu:

*gotoAndPlay(7);*

*gotoAndPlay(20);*

*gotoAndPlay( my frame label);*

Takođe je moguće posebno naznačiti i scenu ukoliko film sadrži više od jedne scene. Ako se akciji gotoAndPlay koristi samo sa jednim parametrom, taj parametar je frejm. Ali. ako se koriste dva parametra prvi parametar je ime scene a drugi ime ili broj frejma.

*gotoAndPlay (My Scene, My frame);*

Kada se izda akcija gotoAndPlay, film prelazi na naznačeni frejm od kog zatim film nastavlja da se prikazuje. Ipak za prelazak sa jednog frejma na drugi i zaustavljanje na tom frejmu koristi se akcija gotoAndStop. Ova akcija postupa na isti način kao i gotoAndPlay, sa istim opcijama vezanim za korišćenje jednog ili dva parametra.

Na isti način, kao i kod prethodne akcije, moguće je kontrolisati unutrašnji Timeline nekog simbola pomoću njegovog Instance Name-a.

Akcija:

*kvadrat.krug.gotoAndPlay(2);*

će na glavnoj sceni pronaći simbol *kvadrat*, u njemu simbol *krug* i u njegovom Timeline-u skočiti u drugi frejm i odatle pustiti film.

**4.3 AKCIJE *nextFrame()* i *prevFrame()***

Ove dve naredbe vrše prelazak na jedan frejm unapred *(nextFrame*) ili jedan frejm

unazad *(prevFrame*) i tu zaustavljaju film. Naime, sintaksa je jednostavna, bez parametara:

*nextFrame();*

*prevFrame();*

Ove akcije se takođe najčešće dodaju uz neki uslov (neku akciju korisnika). Akcijama je moguće kontrolisati i Timeline nekih simbola na sceni, upotrebom putanje:

*kvadrat.nextFrame();*

*krug.prevFrame();*

**4.4 AKCIJE *nextScene()* i *prevScene()***

Kao i kod prethodnih naredbi, ove dve akcije vrše skok na sledeću, odnosno prethodnu scenu našeg Flash filma (ukoliko one postoje). Sintaksa je jednostavna:

*nextScene();*

*prevScene();*

Ovu akciju možete postaviti unutar nekog uslova kao reakciju na klik ili je možete postavitine neki frejm tako da se akcija aktivira čim glava za čitanje dođe do tog frejma.

# 5. USLOVNE AKCIJE

Osim akcija koje se dešavaju u okviru filma kada glava za čitanje dođe do određenog frejma, moguće je postaviti i akcije koje će se izvršavati pri određenoj akciji korisnika ili pri nekom uslovu. Naime, često će nam biti potrebno da zaustavimo film nakon klika korisnika ili da određeni deo filma prikažemo kada korisnik pređe mišem preko nekog simbola.

Takve akcije dodajemo određenim simbolima, uz ispitivanje aktivnosti korisnika koja jedovela do akcije ili aktivnosti simbola na sceni.

Najvažnije akcije iz ove grupe su:

* *on();*
* *if…else*
* *switch…case*
* *for...while*

**5.1 AKCIJA *on();***

Ovu akciju ćemo koristiti ako želimo da pokrenemo akciju (ili više akcija) na određenu aktivnost korisnika. U praksi, to znači da ćemo ovu akciju koristiti kada želimo da Flash film reaguje kada korisnik klikne na neki simbol ili pređe kursorom miša preko nekog simbola. Sintaksa akcije je:

*on(događaj) {*

*akcija;*

*}*

Dakle, za neki događaj, izvršava se akcija koja se nalazi između vitičastih zagrada.

Događaji mogu biti:

* **rollOver** – prelazak kursora preko simbola,
* **rollOut** – sklanjanje kursora sa simbola,
* **press** – pritisak tastera miša dok je kursor iznad simbola,
* **release** – puštanje tastera miša dok je kursor iznad simbola,
* **releaseOutside** – puštanje tastera miša dok je kursor van granica simbola, a bio je pritisnut dok je kursor bio iznad simbola,
* **dragOut** – pritisnut je taster miša dok je kursor bio iznad simbola, a zatim je pomeren kursor van granica simbola (bez puštanja tastera),
* **dragOver** – pritisnut je taster miša dok je kursor bio iznad simbola, zatim je kursor pomeren van granica simbola, a zatim ponovo vraćen,
* **keyPress**(*taster*) – pritisnut je određeni taster na tastaturi.

Što se akcija tiče, to mogu biti bilo koje akcije koje se mogu dodeliti u okviru Actionscript-a. Primer:

*on(release) {*

*\_root.gotoAndPlay(20);*

*}*

Ovako dodata akcija nekom simbolu, znači da će se na klik na taj simbol glava za čitanje pomeriti na 20. frejm i pustiti film odatle. Moguće je dodati i više akcija:

*on(rollOver) {*

*\_root.kvadrat.stop();*

*.*

*.*

*.*

*\_root.gotoAndPlay(20);*

*}*

**Primer:**

Kreiraćemo animaciju kvadrata na glavnoj sceni. U posebnom sloju kreirajte novi simbol kojem ćete dodeliti akciju:

*on(press)*

*{*

*\_root.stop();*

*}*

*on(release){*

*\_root.play();*

*}*

Ako testirate film, animacija će se videti sve do trenutka dok ne kliknete na simbol (i zadržite pritisnut taster). Tada se ona zaustavlja, a nakon puštanja tastera miša ponovo kreće od mesta na kojem je zaustavljena.

Drugi način za dodavanje ove akcije je putem funkcije. Naime, ako sada sklonite akcije sa simbola koje ste uneli u primeru, i tom simbolu dodelite Instance Name *taster,* možete dodati akcije ovom simbolu, ali direktno na frejmu. Dakle, sada ne treba da selektujete simbol, već selektujete frejm u kojem se nalazi simbol, a zatim upišete kod:

*taster.onPress = function(){*

*\_root.stop();*

*}*

*taster.onRelease = function(){*

*\_root.play();*

*}*

Sada smo na Timeline-u dodali akciju koja se odnosi na simbol sa Instance Name-om *taster* i na određenu aktivnost pozivamo funkciju.

Koncept je isti kao u prethodnom načinu, jedina razlika je u sintaksi i u načinu postavljanja koda.

Osim *onPress* i *onRelease* događaja postoje još i *onRollOver* i *onRollOut* uslovi koji se javljaju na prelazak kursora miša preko simbola i na sklanjanje kursora miša sa simbola. Obratite pažnju na velika i mala slova u ovim rezervisanim rečima. Akcija neće raditi ukoliko ne napišete reč ispravno.

**5.2 AKCIJA *if...else***

Uslovni iskaz očekuje od skripta da odredi da li je nešto tačno (*True*) ili netačno (*False*) i da deluje u zavisnosti od toga. Uslovni iskazi imaju široku primenu. Na primer, da odrede da li se neki video klip učitao, da proceni unos korisnika kao što je šifra, odgovori na pitanja iz ankete.

Najčešće korišćeni uslovni iskazi su *if* iskazi koji proveravaju vrednost ili izraz u zagradi. Ako je vrednost *True*, linije koda u vitičastim zagradama se izvršavaju, a ako je vrednost *False*, ActionScript ih ignoriše. Često se u ovakve iskaze dodaje *Else* iskaz koji nudi alternativne instrukcije ako uslov nije *True*.

Nakon toga se u vitičastim zagradama dodaje metod, kako bi se skriptu dale instrukcije.

Sintaksa akcije je:

*if (uslov) {*

*akcija1;*

*akcija2;*

*…*

*akcija10;*

*}*

Dakle, ako je ispunjen neki uslov, izvršiće se akcije između otvorene i zatvorene vitičaste zagrade. Pored toga, može se dodati i set akcija koje će se izvršiti ako nije ispunjen uslov:

*if(uslov) {*

*akcija1;*

*akcija2;*

*…*

*akcija10;*

*}*

*else {*

*akcija11;*

*akcija12;*

*…*

*akcija20;*

*}*

Ako se ispuni uslov, izvršiće se akcije od broja 1 do broja 10, a ako nije ispunjen uslov, izvršiće se akcije od broja 11 do broja 20.

Recimo da imamo promenljivu *username* i želimo da je ispitamo. Kao primer imamo sledeći kod:

*if(username=="korisnik") {*

*\_root.gotoAndPlay("prihvacen");*

*}*

*else {*

*\_root.gotoAndPlay("odbijen");*

*}*

Ako promenljiva *username* ima vrednost *"korisnik"*, glava za čitanje skače na frejm „prihvaćen“ i odatle pušta film. U tom delu treba dati ispis da je korisničko ime prihvaćeno.

Ukoliko uneto korisničko ime ne odgovara vrednosti „korisnik“, glava za čitanje se pomerana frejm „odbijen“ i odatle pušta film.

Ukoliko je potrebno postaviti uslov iz dva dela (ili više njih) i to je moguće upotrebom *if*akcije. Naime, ako vam je potrebna provera ne samo korisničkog imena, već i lozinke, uslovmožete postaviti ovako:

*if((username=="korisnik")&&(password=="b3KE77a")) {*

*\_root.gotoAndPlay("prihvacen");*

*}*

*else {*

*\_root.gotoAndPlay("odbijen");*

*}*

Uslov će sadržati dva dela. U prvom delu se ispituje da li promenljiva *username* ima vrednost „korisnik“, a u drugom delu da li promenljiva *password* ima vrednost „b3KE77a“. Ukoliko su oba uslova ispunjena (i samo tada), izvršiće se akcija.

Da bismo naznačili da je potrebno da oba uslova budu ispunjena, koristimo &&. To je zamena za AND odnosno oznaku da je potrebno ispuniti oba uslova. Ukoliko želimo da dodamo OR tako da bude ispunjen barem jedan uslov, umesto te oznake koristićemo ||:

*if((username=="korisnik")||(password=="b3KE77a")) {*

*\_root.gotoAndPlay("prihvacen");*

*}*

*else {*

*\_root.gotoAndPlay("odbijen");*

*}*

Umesto *else* dela, moguće je postaviti *else if* kod, koji u slučaju da prvi uslov nije ispunjen, postavlja drugi uslov i daje set akcija u tom slučaju:

*if(username=="korisnik") {*

*\_root.gotoAndPlay(“prihvacen”);*

*}*

*else if (username=="admin") {*

*\_root.gotoAndPlay("admindeo");*

*}*

*else {*

*\_root.gotoAndPlay("odbijen");*

*}*

Ovde imamo tri segmenta: najpre se ispituje da li je *username* "korisnik" (i šalje glavu za čitanje na frejm "prihvaćen"), a ako nije, ispituje da li je vrednost promenljive *username* "admin" (i šalje glavu za čitanje na frejm "admindeo"). Ako nijedan uslov nije ispunjen, glava za čitanje se šalje na frejm "odbijen".

Međutim, za situacije kada imate dosta *else if* segmenata, bolje je koristiti sledeću akciju.

**5.3 AKCIJA *switch…case***

Ova akcija je takođe uslovna akcija i ispituje neku vrednost. U zavisnosti od rezultata izvršava neku akciju. Ova akcija je zgodna ukoliko postoji više mogućih rezultata od kojih svaki uslov proizvodi određenu akciju. Sintaksa je:

*switch (promenljiva) {*

*case vrednost1:*

*akcija1;*

*break;*

*case vrednost2:*

*akcija2;*

*break;*

*case default:*

*akcija3;*

*}*

Dakle, za svaku moguću vrednost promenljive iz uslova definišemo koje će se akcije izvršiti. Ukoliko se nijedan od uslova ne ispuni, u delu *default* definišemo koje će se akcije izvršiti. Recimo, za malopređašnji primer, možemo definisati ovakvu akciju:

*switch (username) {*

*case "korisnik":*

*\_root.gotoAndPlay("prihvacen");*

*break;*

*case "admin":*

*\_root.gotoAndPlay("admindeo");*

*break;*

*default:*

*\_root.gotoAndPlay("odbijen");*

*}*

**5.4 AKCIJA *for***

Sintaksa za kreiranje petlji u ActionScriptu je malo složenija od jednostavnih if iskaza. U ActionScriptu petlje se kreiraju na isti način kao i u C, C++ i Java jezicima.

Osnovni tip petlje je petlja for

*for (var i =0; i<10; i++) {*

*trace(i);*

*}*

Ako pokrenete ovaj kod u skripti koja je pridružena nekom frejmu, dobićete brojeve od 0 do 9 smeštene u prozor Output. Petlja broji od 0 do 9, menjajući usput lokalne varijable.

Kao što se može videti, iskaz for ima tri dela. Svaki deo je odvojen znakom;

Prvi deo je deklaracije promenljive. U ovom slučaju lokalna promenljiva i je kreirana i podešena na 0. Ovaj prvi deo for iskaza izvršava se samo jednom, pre nego što petlja startuje.

Drugi deo for iskaza je uslov. U ovom slučaju ovaj deo proverava da li je i manje od 10. Petlja for nastavlja da se vrti sve dotle dok je ovaj uslov tačan. Kada se petlja startuje i je jednako 0, koje je svakako manje od 10, tako da se petlja startuje.

Treći deo iskaza for je operacije koje se izvrši svaki put kada petlja vrti. U ovom slučaju i je povećan za 1. Ova operacija se izvršava nakon svakog povećanja petlje zato što je operator ++ smešten na i. Kada bi ovaj operator bio postavljen pre i, kao ++i, ova operacija bi se obavljala, pre nego što bi naredbe u petlji bile izvršene.

**5.5 DELOVI SKRIPTI**

Prilikom pisanja skripte, koriste se sve vrste različitih reči i znakova. Da bi se ovo naučilo, bilo bi korisno pogledati u neku pravu skriptu i ispitati njene delove.

Koristićemo sledeću skriptu kao primer. To je skripta za dugme, koja se startuje kada korisnik klikne na dugme kojem je skripta pridružena. Ona ne izvršava neku određenu funkciju, već više demonstrira nekoliko osnovnih sklopova ActionScripta.

on (press){

var myVariable = 7;

var myOtherVariable = “Macromedia”;

for (var i=0; i<10; i++){

trace (i);

if (myVariable + 3 ==5){

trace (myOtherVariable);

}

}

}

Prva linija u skripti određuje ostatak kao nešto što se izvršava kada korisnik prvi put klikne na dugme. Sklop on (press) može se koristiti samo u skripti za dugme. Takođe se može koristiti on(release) ako želite da se kod izvršava kada korisnik završi sa pritiskom na dugme.

Velika zagrada, { , na kraju prvog reda označava da je to početak segmenta koda. Kojd koji se nalazi između ove otvorene i njemu odgovarajuće zatvorene zagrade je deo seta koda koji je jedna celina. U ovom slučaju, taj segment koda predstavlja ceo kod koji se izvršava priliko pritiska na dugme.

Prvi red ovog segmenta koda pravi novu lokalnu promenljivu nazvanu myVariable i dodeljuje mu vrednost 7. Novi red dodeljuje string Macromedia promenljivoj myOtherVariable. Tačka-zarez ; na kraju ovog reda, i na kraju većine ostalih redova označava da je naredna uobličena.

Konstrukcija for startuje pretlju. U ovom slučaju, petlja se ponavlja 10 puta, sa promenljivom i, koja se startuje od 0 i ide do 9. Zagrada na kraju reda for označava početak segmenta koda koji će služiti kao telo petlje.

Naredba trace šalje sadržaj koda na prozor Output. Naredba if je uslovni iskaz. Ona testira iskaz koji joj sledi, myVariable + 3 == 5, da bi proverila da li je tačan. Ako jeste, izvršava segment koda u zagradama. Ako nije, onda se preskače ceo segment koda. U pitanju je jednostavna naredba trace koja šalje sadržaj promenljive myVariable na proyor Output. Ostatak koda se sastoji iz 3 zatvorene zagrade. Prva zatvara iskaz if , druga zatvara petlju i treća segment on (press).

# 6. KOMUNIKACIJA SA SPOLJNIM FAJLOVIMA

Izrada Flash animacija i aplikacija često podrazumeva upotrebu nekih drugih fajlova koji se nalaze na lokalnom računaru ili negde na Internetu. Komunikacija može biti u vidu linka ka nekoj internet strani, može se zasnivati na učitavanju promenljivih iz nekog .txt fajla i može predstavljati „uvlačenje“ nekog eksternog SWF fajla u glavni film.

Akcije koje će nam biti potrebne su:

* getURL();
* loadVariable();
* loadMovie();

**6.1 AKCIJA *getURL()***

Ukoliko je potrebno skočiti na određeni sajt sa vašeg sajta, koristićemo ovu komandu. Naime, ponekad je potrebno na sajtu imati linkove koji će na klik otvarati neki novi sajt ili neku stranicu na sajtu. Ovu akciju možete dodati i simbolu na sceni uz *on()* uslov a moguće je i postaviti direktno na frejm.

Kreiramo simbol i dodajemo mu akciju:

*on(press){  
     getURL("http://www.adresa.com/strana.html" , "\_blank", "GET");*

*}*

Dakle, tri parametra ove akcije su:

* adresa na kojoj se nalazi strana,
* prozor u kojem će se ta strana otvoriti,
* način na koji će promenljive tog filma biti prosleđene strani.

Poslednji parametar je opcioni i on se koristi samo kada želimo da prosledimo promenljive filma nekoj strani koja će te podatke procesuirati. Ukoliko želimo jednostavan link, nije neophodno stavljati ovaj parametar.

Ukoliko se koristi \_blank, linkovana strana će se otvoriti u novom prozoru, dok ukoliko stavimo \_self, nova strana će se otvoriti u istom prozoru Browser-a.

**6.2 AKCIJA *loadVariables***

Ukoliko poželite da iz eksternog fajla (.txt formata) uvezete promenljivu u Flash film, to možete uraditi akcijom loadVariables. Ovom akcijom možete učitati promenljive iz teksta koje je generisao i neki CGI ili Perl skript, kao i ASP i PHP strana koja može da komunicira i sa SQL bazom podataka.

Sintaksa je sledeća:

*loadVariables ("url", cilj, "način\_slanja/primanja\_varijabli");*

Parametri su sledeći:

• URL predstavlja adresu fajla sa kojim je potrebno razmeniti promenljive,

• cilj predstavlja nivo ili ime instance koja će primiti varijable,

• način slanja/primanja se odnosi na GET ili POST metod.

Kao primer, možemo dodati kod:

*loadVariables ("podaci.txt", \_root.podaci, "POST");*

Na ovaj način ćemo učitati promenljive iz fajla podaci.txt i učitati ih u simbol sa InstanceName-om „podaci“ koji se nalazi na glavnoj sceni.

**Primer:**

1 . Kreirajte novi Flash film i u njemu napravite Dynamic text box, a zatim u polju

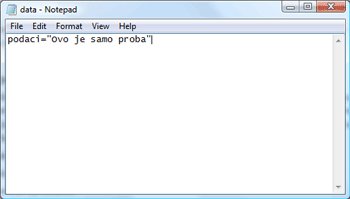
Variables u Properties Inspector-u upišite *podaci*. Snimite taj FLA fajl u neki

direktorijum na hard disku.



2. Kreirajte txt fajl u istom folderu gde se nalazi FLA fajl i u njemu upišite:

*podaci="Ovo je samo proba"*



3. U prvom frejmu glavnog Timeline-a dodajte akciju:

*loadVariables("data.txt",\_root);*

Ako testirate film, u dinamičkom tekst okviru bi trebalo da se pojavi tekst "Ovo je samo proba".

Naime, akcija koju smo upisali poziva fajl "data.txt" i učitava sve varijable u glavni Timeline (zato što smo kao parametar stavili ***\_root***). Pošto se u txt faju nalazi varijabla *podaci* sa sadržajem "Ovo je samo proba", ona se učitava u Flash film. Kako smo dinamičkom tekst okviru dodelili vrednost promenljive *podaci*, tekst se automatski prikazuje u tom boksu.

U txt fajl smo mogli da dodamo još promenljivih:

*podaci="prvi tekst"&email="email@adresa.com"&adresa="postanska adresa"*

Pošto su sve promenljive razgraničene znakom AND (&) svi će biti učitani i možemo ih upotrebiti u Flash filmu. Da bismo upotrebili i ove nove promenljive, moraćemo dodati još dva tekst okvira Dynamic tipa i za svako polje definisati Variable (drugi okvir će imati Variable ***email****,* a treći okvir Variable ***adresa***).

Ukoliko vam je potrebno da pošaljete podatke nekom PHP skriptu (kao na primer podatke iz kontakt formulara), za adresu fajla stavite neki PHP fajl, a za metod slanja izaberite *GET* ili *POST*. U samom PHP fajlu, u zavisnosti od izabranog metoda, morate prihvatiti te promenljive, a zatim ih procesuirati.

**6.3 AKCIJA *loadMovie();***

U Flash-u kreiramo simbole i animacije koje se pri izvozu nalaze u jednom SWF fajlu. Međutim, moguće je u jedan SWF fajl u toku emitovanja „uvući“ drugi SWF fajl i prikazati ga kao deo glavnog filma.

Akcija koja služi za to je loadMovie(). Njom je moguće u neki postojeći simbol na sceni učitati neki eksterni SWF fajl i prikazati ga.

Ako, na primer, imamo simbol na glavnoj sceni koji ima X,Y koordinate 0,0 i Instance Name *placeholder* i ako dodamo akciju na glavni Timeline:

*loadMovie("dodatni\_fajl.swf", placeholder);*

kada glava za čitanje dođe do tog frejma, umesto simbola *placeholder* pojaviće se film iz fajla *dodatni\_fajl.swf*.

Mogli smo da dodamo i apsolutnu putanju do simbola:

*loadMovie("dodatni\_fajl.swf", \_root.placeholder);*

i rezultat bi bio isti. Fajl dodatni\_fajl.swf mora biti u istom folderu kao i fajl koji ga „uvlači”, a ako nisu u istom folderu, onda je potrebno naznačiti putanju do dodatnog fajla:

*loadMovie("flash/files/dodatni\_fajl.swf", \_root.placeholder);*

Na ovaj način možete uvlačiti i JPG slike, a upotrebom Actionscript 3.0 jezika možete pravitii mnogo naprednije kombinacije fajlova.

# 7. KORISNIČKE FUNKCIJE i *onClipEvent()*

**7.1 FUNKCIJE**

Funkcija je niz naredbi koje čine jednu funkcionalnu celinu. Funkcije nam omogućavaju da niz naredbi koje se često ponavljaju u našem programu stavimo u jednu celinu i koristimo po potrebi. Na primer, ako u našem programu imamo potrebu za izračunavanjem kvadrata nekog broja, onda je puno bolja opcija da kod koji izvršava tu matematičku operaciju stavimo u jednu funkciju koju onda po potrebi pozivamo, nego da nam se taj kod ponavlja u programu na više različitih mesta. Time se dobija manji i efikasniji program:

*function myFunction (num){*

*var newNum = num +3;*

*return newNum;*

*}*

Funkcija počinje sa ključnom reči function iza koje sledi ime funkcije. Ime fukncije može biti bilo šta, kao što je to slučaj i sa promenljivim. Ali bi ta imena trebalo davati prema osnovnoj nameni fukncija.

Nakon imena fukncije dolazi leva velika zagrada. Zati sledi lista parametara. Parametar je promenljiva koja je definisana u trenutku kada je funkcija pozvana. Mislite o ovome kao o ulaznim podacima fukncije. U ovom slučaju funkciji ćemo dodeliti broj sa kojim ona treba nešto da uradi.

Funkcija može imati jedan, više ili nijedan parametar. U svakom slučaju sekciju sa parametrima zatvarate desnim velikim zagradama, a zatim koristite otvorenu zagradu za startovanje fukncije.

Svi redovi između otvorenih i zatvorenih zagrada su naredbe u okviru funkcije. U ovom slučaju, kreirana je nova lokalna promenljiva pod imenom newNum. Vrednost newNum je podešena tako da se bez obzira šta je vrednost promenljive num, povećava za 3. Tako ako u fuknciju unesete 7 za num, newNum će biti 10.

Komanda return je posebna komanda koja se jedino koristi u funkcijama. Ova komanda kompletira fuknciju i određuje vrednost kao rezultat funkcije.

Ako umesto prethodne funkcije upišemo:

*function Kvadrat(a:Number){*

*trace(a\*a);*

*}*

*Kvadrat(10);*

mi pozivamo funkciju prosleđujući joj parametar (broj 10). Funkcija prihvata taj parametar,a zatim vrši ispis kvadrata tog broja. Deo *a:Number* predstavlja definiciju tipa te promenljive (u našem slučaju je tip broj).Ako sada proširimo priču još dalje, možemo napisati ovakvu funkciju:

*function Kvadrat(a:Number){*

*broj=a\*a;*

*return(broj);*

*}*

*rezultat=Kvadrat(10);*

*trace(rezultat);*

Ovde uvodimo nove promenljive. Dakle, najpre imamo promenljivu *rezultat* koja predstavlja krajnji rezultat koji će biti ispisan komandom *trace*. Toj promenljivoj dodajemo vrednost funkcije Kvadrat(10). Kada pozovemo tu funkciju, parametar 10 se prosleđuje funkciji koja u sebi ima novu promenljivu *broj*.

Vrednost promenljive *broj* je jednaka kvadratu prosleđenog parametra. Funkcija onda vraćanazad tu vrednost koja se sada dodeljuje promenljivoj *rezultat.*

Možda vam ovo sada deluje konfuzno, ali vrlo brzo ćete shvatiti princip i koristiti funkcije svakodnevno.

U okviru funkcije Kvadrat smo uveli varijablu *broj.* Međutim, nismo je definisali rečju *var* iz prostog razloga što bi onda ona postala lokalna promenljiva koja bi važila samo u okviru te funkcije. Nama je potrebno da se vrednost te promenljive prosledi van funkcije (i dodeli promenljivoj *rezultat*).

Zadnja dva reda:

*rezultat=Kvadrat(10);*

*trace(rezultat);*

možemo da skratimo u jedan:

*trace(Kvadrat(10));*

Na ovaj način izbacujemo promenljivu *rezultat* uz isti efekat. Takođe, i linije:

*broj=a\*a;*

*return(broj);*

možemo da skratimo na:

*return(a\*a);*

i time izbacimo i promenljivu *broj*.

Na ovaj način smo značajno pojednostavili kod, a jedini razlog zašto smo krenuli od komplikovanijeg načina je kako biste se upoznali sa svim mogućnostima funkcija.

*function Kvadrat(a:Number)*

*{*

*return(a\*a);*

*}*

*trace(Kvadrat(10));*

Osim ovakvih proračuna, funkcije možete koristiti za mnogo naprednije akcije koje se često ponavljaju (i to na različitim mestima u filmu). Jednom definisana akcija je prisutna svuda i možete je pozvati kada god to poželite.

**7.2 *onClipEvent()***

Kao što smo već videli, upotrebom uslova on() možemo dodeliti neke akcije koje se izvršavaju na prelazak kursora miša preko simbola, ili na klik. Pored tog uslova, imamo još jedan, koji se takođe dodaje simbolu, samo su događaji drugačiji:

* ***enterFrame*** *–* akcije se izvršavaju kada god se simbol pojavi na sceni u nekom frejmu. U praksi to znači da se akcije izvršavaju onoliko puta u sekundi koliki je Frame rate tog filma,
* ***load*** *–* akcije će se izvršiti samo kada se simbol pojavi na sceni (učita u filmu),
* ***unload*** – akcije će se izvršiti samo kada se simbol *ukloni*, tj. u prvom frejmu nakonsklanjanja simbola sa scene iz filma,
* ***mouseMove*** – akcije će se izvršiti na svako pomeranje kursora miša.

Recimo, ako želimo da se neki simbol pomera za po 30 px u sekundi (ako je Frame Rate 30 FPS), ali postepeno, piksel po piksel, možemo nekom simbolu dodati ovaj kod:

*onClipEvent(enterFrame){*

*imesimbola.\_x++;*

*}*

Na ovaj način smo u svakom frejmu pomerili vrednost X simbola *imesimbola* za po 1 podeok.

**Primer:**

1. Kreirajte simbol na sceni (kvadrat) i dodelite mu Instance Name *kvadrat*. Smestite

ga uz levu ivicu radne površine.

2. Selektujte simbol, a zatim u Actions panelu dodajte kod:

*onClipEvent(enterFrame){*

*\_root.kvadrat.\_x++;*

*}*

Na ovaj način, svako pojavljivanje simbola u frejmu će pokrenuti akciju koja će X koordinatu simbola povećati za 1 podeok. Možemo uneti i ovakav kod i dobićemo isti rezultat:

*onClipEvent(enterFrame){*

*this.\_x++;*

*}*

Rezervisana reč ***this*** služi da označi kojem simbolu ćemo menjati X koordinatu, a pošto smoakciju dodali simbolu, odnosiće se na taj simbol. Ukoliko želimo da pomeramo simbol po X koordinati za više od 1 px, možemo upisati sledeći kod:

*onClipEvent(enterFrame){*

*this.\_x += 5;*

*}*

Dakle, u svakom frejmu, vrednosti X koordinate će biti dodato 5 jedinica (zbog operatora +=). Ukoliko umesto *enterFrame* stavimo *mouseMove*, simbol će se pomerati samo u slučaju da pomeramo kursor miša po radnoj površini:

*onClipEvent(mouseMove){*

*this.\_x++;*

*}*

Takođe, umesto akcije za pomeranje simbola po x osi, možemo staviti bilo koju akciju. Ova akcija je stavljena samo da pokažemo kako se mogu kreirati tranzicije ovom akcijom. Takođe, uopšte nije neophodno menjati sam simbol ovom akcijom. Akcija ispituje sam simbol, ali može uticati i na druge simbole na sceni.

Tako recimo, možemo napraviti Preloader gde ćemo ovu akciju dodati nekom simbolu na sceni (bilo kojem) uz događaj *enterFrame*, a kreirati set akcija koji će računati procenat učitanog dela Flash fajla i koji će prikazivati taj procenat u vidu dinamičkog teksta.

*onClipEvent (enterFrame) {*

*\_parent.loading = \_parent.getBytesLoaded();*

*\_parent.total = \_parent.getBytesTotal();*

*\_parent.percent = (\_parent.loading/\_parent.total)\*100;*

*\_parent.per=int(\_parent.percent);*

*}*

Sada je potrebno kreirati dinamički tekst boks koji će u polju Variable imati upisanu promenljivu *per*. Sa svakim učitavanjem simbola na sceni (u svakom frejmu) vršiće se kalkulacija i menjaće se vrednost promenljive *per*, a ona će biti upisivana u tekst boksu.

# 8.PREUZIMANJE PODATAKA OD KORISNIKA

**8.1 OTKRIVANJE PRITISKA NA TASTERE**

Postoje tri načina za otkrivanje jednostrukih pritisaka na tastere tastature. Prvi koristi dugmad.Drugi, ključne objekte a treći koristi prisluškivače.

**8.2 DETEKCIJA PRITISKA POMOĆU TASTERA**

Dugmad se mogu koristiti za detekciju pritiska na tastere tastature. Sve što je potrebno je korišćenje posebnog manipulatora onu skripti koja je pridružena dugmetu. Na primer, imamo manipulator koji istražuje pritisak tastera a:

on (keyPress “a”) {

trace (“ Key ‘a’ pressed.”);

}

Manipulator *on(keyPress)* je veoma osetljiv na velika i mala slova, tako da onaj koji detektuje malo aneće detektovati i veliko A*.* Zato je potrebno koristiti poseban  *on(keyPress)* manipulator za veliko slovo A*.* Na sreću, skripte koje se pridružuju dugmadima mogu sadržati onoliko manipulatora koliko želimo.

Za detekciju nekih drugih tastera, kao što su recimo tasteri sa strelicom, koriste se posebni kodovi. Na primer, *<Left>* se može koristiti za detekciju tastera sa strelicom na levo.

*on (keyPress “<Left>”) {*

*trace (“Left pressed.”);*

*}*

Takođe se mogu kombinovati i događaji na iskazu *on*. Na primer, pretpostavimo da želite dugme koje ima prečicu sa tastature. Normalno je da uradite jedan *on(release)* manipulator. Međutim, možete uraditi jedan neznatno kompleksniji manipulator koji će upravljati i klikom i pritiskom na taster u isto vreme:

*on (keyPress “b”, release) {*

*trace (“ ‘b’ pressed or buton clicked. “);*

*}*

**8.3 Key OBJEKT**

Mada su dugma korisna za razumevanje jednostrukog klika, ipak nisu baš najbolja rešenja za detektovanje na primer, da li se taster drži pritisnut, recimo ako želite da napravite igricu u kojoj se igrač kreće sve dok je taster pritisnut.

Za ovaj tip funkcionisanja dugmadi potreban je objekt *Key*. Objekt *Key* je set funkcija i konstanti koji je ugrađen u Flash. Ove funkcije i konstante se mogu koristiti za određivanje da li je taster pritisnut. Na primer, da biste videli da li je taster pritisnut sa strelicom na levo koristi se sledeći kod:

*if (Key.isDown(Key.Left)) {*

*trace(“The left arrow is down”);*

*}*

Funkcija *Key.isDown* vraća rezultat tačno ili netačno u zavisnosti od toga da li je taster koji je naveden kao parametar pritisnut. Konstanta *Key.Left* predstavlja taster sa strelicom na levo. Prema tome, iskaz if je tačan u slučaju da je taj taster pritisnut.

Ali šta ako želite da detektujete običan taster kao što je *a*, na primer? Još uvek je potrebno da znate kod za taj taster, ali ne možete koristiti znak *a*. To se postiže funkcijom *Key.getCode*. Na primer:

*if (Key.isDown(Key.getCode(“a”)) {*

*trace(“The left arrow is down”);*

*}*

**8.4 Key Listeners**

Problem sa metodom detektovanja tastera pomoću objekta *Key* je što se pomoću njega ne mogu detektovati inicijalni pritisci na taster. Pomoću nje se samo testira, iznova i oznova, da li je neki taster pritisnut.

Manipulator *onClipEvent(EnterFrame)* se pokreće i proverava da li je neki taster pritisnut. Nije. Zatim korisnik klikne brzo pre no što se sledeći put startuje *onClipEvent(EnterFrame)* manipulator. Kada se ovaj manipulator startuje, taster više nije pritisnut. I taj pritisak nikada ne može biti prepoznat od strane manipulatora. U slučaju da korisnik brzo pritiska, Flash film će propustiti da registruje pritisak.

Još jedan problem se pojavljuje u slučaju da korisnik želi da klikne neoliko puta. Na primer, pritiskanjem razmaka se može ispaliti metak u igrici. Objekt *Key* možda ne može napraviti razliku između jednog ili drugog pritiska i serije nekoliko brzih.

Postoje tri mogućnosti izbora prisluškivanja kada dolazi do prisluškivanja pritiska na taster.

Možete kreirati osluškivač koji će obratiti pažnju koji se događaj na tasteru dešava. Osluškivač se sastoji od dva dela. Prvi deo je njegovo kreiranje. Morate reći filmu da želite da osluškujete neki događaj. U ovom slučaju, to ćete učiniti sledećom naredbom:

*Key.addListener (\_root);*

Naredba *Key.addListener(\_root);* uzima neki objekt, kao što je izvor filma ili neki klip kao svoj parametar. To je objekat koji prima poruku kada se taj događaj desi.

U ovom slučaju izvorni nivo filma preuzima događaj. Da bi izvorni nivo mogao odreagovati na ovaj događaj, mora sadržati funkciju koja mu je dodeljena da bi to moga ostvariti.

Osluškivač tastera može slati dve vrste poruka: *onKeyUP* i *onKeyDown*. Da biste napravili funkciju koja manipuliše jednom od ovih poruka, morate napisati kod na sledeći način:

*\_root.onKeyUp =function() {*

*trace(Key.getAscii());*

*}*

Umesto \_root možete koristiti i neki drugi objekat, kao što je recimo klip pod uslovom da je u putanju isti objekat koji je korišćen u naredbi Key.addListener.

**8.5 UPIS I ISPIS TEKSTA *(Input Text)* i *(Dynamic Text)***

Kada se pomene upis teksta, najpre se setimo kupovine preko Interneta. Prilikom kupovine preko Interneta potrebno je da posetilac upiše ime, adresu i podatke sa kreditne kartice. Sve to se upisuje u tekstualna polja koja se izrađuju bilo u HTML-u ili u Flash-u. Iz polja za unos teksta koja napravimo u Flash-u možemo prikupiti informacije od posetilaca i obraditi ih u Flash-u (pomoću ActionScript-a) ili tako što ćemo ih poslati nekoj aplikaciji na serveru da ih obradi.

Polje za upis teksta pravi se u Flash-u pomoću alatke Text. Primenjujemo sličan postupak kao kad pravimo dinamički i statički tekst. Podesimo nekoliko opcija u Property Inspector-u i posetioci će moći da upisuju tekst. Tekstu pristupamo pomoću ActionScript-a.

Osnovni koncept je takav da i polja za unos teksta (*Input Text*) i dinamička polja (*Dynamic Text*) imaju promenljivu (varijablu) vezanu za sebe. Kod *Input* teksta, vrednost te promenljive se upisuje od strane korisnika, dok *Dynamic* tekst služi samo za ispis vrednosti promenljive.

Uloga Actionscript-a je da vrši procesuiranje unetih vrednosti varijable u *Input* tekst polju i da obrađenu varijablu pošalje u *Dynamic* tekst polje. Akcije se mogu dodeliti na neku aktivnost korisnika (klik na neko dugme) ili se jednostavno mogu dešavati stalno u pozadini.

Ako imamo polje koje je *Input* tekst tipa i dodelimo mu Instance Name *polje1*. Sadržaju tog polja možemo pristupiti na dva načina:

1. preko promenljive upisane u polje Variable – ako se promenljiva zove *prom*, sadržaj

Input tekst polja će automatski biti dodeljen toj varijabli,

2. preko sadržaja tekst polja – upotrebom koda *polje1.text* možemo doći do upisanog

teksta u tom polju (npr. ako imamo promenljivu *mojTekst* i unesemo kod *mojTekst*

*= polje1.text* ta promenljiva će preuzeti tekst koji je upisan u to polje.

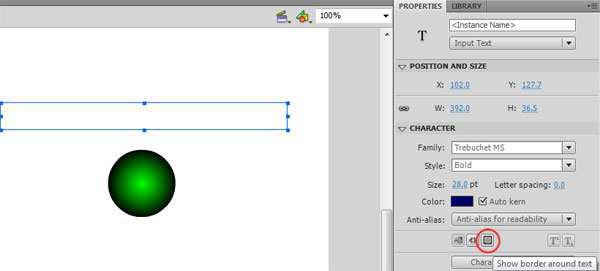
Na ovaj način možemo pristupiti sadržaju polja *Input* text tipa. Na isti način možemo vršiti upis teksta u dinamičko tekst polje. Dakle, ako imamo polje sa Instance Name-om *polje2* i koje je *Dynamic* text tipa, ispis promenljive *mojTekst* možemo izvršiti tako što unesemo kod:

*polje2.text = mojTekst;*

**Primer:**

1. Prvo na sceni nacrtamo, na primer, krug i pretvorimo ga u simbol tipa Button (F8).

2. Pomoću alatke *Text* dodamo na scenu tekstualni element, kao što se vidi na slici. U Property Inspector-u izaberemo font, veličinu slova, boju... Dok je i dalje izabran tekstualni element, odaberemo *Input Text* iz padajuće liste na panou *Property Inspector* i uključimo sličicu *Show border around text*.



3.Prirtisnemo polje Instance Name i upišemo *poljeUpis*. U polje Var upišemo na primer *Upis*. Na ime ove promenljive pozvaćemo se u akciji koju ćemo sada definisati.

4.Pritisnemo alatku Selection pa na sceni izaberemo dugme. Zatim odaberemo

***Window > Actions***.

5. I u panou Actions upišemo sledeći ActionScript kod:

*on (release, keyPress "<Enter>" ) {*

*if (poljeUpis.text != undefined && poljeUpis.text != "") {*

*Ispis = "Upisali ste :" + Upis;*

*}*

*else {*

*Ispis = "Nema upisa.";*

*}*

*}*

6. Sada je potrebno postaviti polje za ispis: Ponovo izaberite *Text tool*, ovog puta

izaberite *Dynamic text*, a u polju *Variable* upišite *Ispis.*

7. Nacrtajte okvir ispod kruga, a zatim selektujte opciju “*Show border around text*”.

Evo šta upisani ActionScript kod govori Flash-u:

Kad neko pritisne i pusti dugme (ili pritisne taster Enter, nakon što unese tekst), proveri koji je tekst upisan u polje sa Instance Name-om *poljeUpis* – ako je uopšte upisan neki tekst. Ako nađe tekst, varijabli *Ispis* dodeljuje vrednost upisanog teksta i ispiše ga na panelu *Output*. Ukoliko ne nađe tekst, evidentira poruku „*Nema upisa*”.

Sada možemo isprobati polja za upis teksta.

1. Izaberemo **Control > Test Movie** (ili *Ctrl+Enter*).

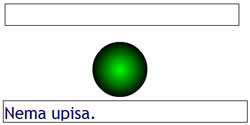
U Flash Player-u koji se pojavljuje vidimo dugme i polje za upis teksta.

2. Pritisnemo dugme.

Flash prikazuje poruku „*Nema upisa*“ u polju za ispis.

3. Upišemo tekst u polje za unos teksta i pritisnemo ili dugme ili taster Enter.

Flash sad evidentira upisani tekst u polju za ispis i pokazuje koji je tekst upisan



**8.6 STRING OPERACIJE**

Za povezivanje dva stringa koristimo simbol +. Na primer, ako imamo promenljivu pod nazivom *myVariable* i želimo nešto da joj dodamo, koristimo jednostavno *myVariable+* i to što želimo da dodamo. Ovo je jedan primer korišćenja prozora *Output*:

*var myVariable=Hello;*

*trace (myVariable + World.);*

Rezultat bi bio “*Hello World*“. Promenljiva još uvek sadrži samo *Hello*, ali naredba *trace* to spaja u “*Hello World“.*

String se takođe može dodati i na kraju promenljive konstantno koristeći +=:

*var myVariable=Hello;*

*myVariable+=World;*

*trace(myVariable);*

**8.7 FUNKCIJE STRINGA**

Postoje nekoliko različitih funkcija stringa, koje spadaju u grupu ostalih ActionScript funkcija. *indexOf* se koristi za pronalaženje pozicije odreženog stringa oji se nalazi u ovkriu nekog drugog stringa. Na primer, da biste pronašli “*W*“, potrebno je da uradite sledeće:

*var myString = Hello World.;*

*trace (myString.indexOf(W, 0));*

Kao rezultat ova funkcija vraća 6, zato što je ”*W*“ šesti karakter. Mogu se pronaći i stringovi duži od jednog karaktera.

*var myString = Hello World.;*

*trace(myString.indexOf(llo, 0);*

Rezultat je 2, zato što string *“llo“* počinje od karaktera 2. Zapamtite da 0 kao drugi parametar ove funkcije znači da pretraga započinje od te tačke. 0 započinje pretragu od pošetka stringa. Ako se na drugi parametar koristi 2, pretraga će ignoristati prva dva karaktera i krenuće tek od trećeg karaktera.

Ako fukncija *indexOf* ne pronađe traženi string, ona se vraća na -1.

Funkcija *lastIndexOf* se koristi za pronalaženje stringova na taj način što pretraga započinje od kraja stringa u okviru koga se nalazi traženi string. Na primer, da biste pronašli prvo slovo “l“ i poslednje “l“, možete uraditi sledeće:

*var MyString = Hello World.;*

*trace (myString.indexOf (l , 0));*

*trace (myString.lastIndexOf ( l ));*

Rezultati su 2 i 9. 2 predstavlja prvo *“l*” u “*Hello”* a 9 predstavlja “l” u “*World”.*

ActionScript, osim toga ima i korisne funkcije koje pretvaraju slova u stringu u velika i mala:

*var myString = Hello World.;*

*trace (myString.toUpperCase());*

*trace(myString.toLowerCase());*

Rezultat u prozoru *Output* će izgledati na sledeći način:

*HELLO WORLD.*

*hello world.*

Kada su u pitanju stringovi potrebno je znati i način dobijanja informacija o njihovoj dužini. U ovu svrhu se koristi atribut length:

*var myString = Hello World.;*

*trace (myString.length);*

Rezultat je 12. Zapamtite da to što brojanje počinje od pozicije 0 uopšte ne utiče na broj karaktera. Naime prvi karakter je 0, a poslednji 11, dok je ukupna dužina 12.

# 9. OBJEKTNO ORJENTISANO PROGRAMIRANJE

Actionscript je vrsta objektno orjentisanog programskog jezika. Zapravo, OOP je nešto što je poznato kao metodologija programiranja, što u osnovi znači da je to samo jedan od načina osmišljavanja i kreiranja rešenja za programerske probleme.

Možemo reći da postoje dve glavne metodologije programiranja: proceduralno i objektno orjentisano programiranje. Najveći broj klasičnih programskih jezuka kao što su Fortran, Pascal i C, zasnivaju se na proceduralnom metodu. To znači da ovi jezici rešavaju probleme u maniru korak po korak. U proceduralnim jezicima određeni deo programa, ako ga često koristite, možete izdvojiti u nešto što se naziva funkcija.

Funkcije nisu ništa drugo no delići programa koji se često koriste. One u stvari ništa ne rade dok ih ne pozovete. Kada pozovete funkciju, možete joj proslediti informacije ili vrednosti koristeći parametre ili argumente. Ona prihvata zadate parametre i vraća rezultate u skladu sa njima.

Mada se proceduralni jezici i dalje upotrebljavaju, oni „pate" od nekih karakterističnih problema. Prvo, iako se funkcije u proceduralnom programiranju mogu koristiti više puta, one ne omogućavaju ponovnu upotrebu većeg dela proceduralnog programa. Zbog toga je najveći deo programa krajnje specijalizovan.

Ova specijalizacija stvara probleme. Ukoliko niste autor programa, možete izgubiti mnogo vremena u pokušajima da zaista razumete šta određeni deo programa radi. Problem se sastoji u tome da morate da razumete skoro svaku liniju programa pojedinačno kako biste razumeli program u celosti.

Sve ovo vodi do krajnjeg problema u vezi sa proceduralnim programiranjem: činjenica je da je sve fokusirano na procedure. U programiranju su podaci ili informacije, podjednako važni kao i sam program. Postoje određene relacije između podataka i programa koje proceduralni jezici ignorišu.

Kao odgovor na pomenute i još neke probleme, stvoreno je objektno orjentisano programiranje kao i mnogi novi jezici: Java, C++, Javascript i Actionscript. Svaki od njih implementira OOP na različite načine, ali ono što je zajedničko za sve jeste to da su bazirani na objektima.

**9.1 OBJEKTI, SVOJSTA I METODI**

Objekat je kolekcija međusobno povezanih podataka i programa. Delovi programa unutar objekta se nazivaju metodi, a delovi podataka ili informacija unutar objekta su svojstva.

Recimo da želite da napravite igru za simulaciju auto-trka. U proceduralnom jeziku morali biste da pazite na mnoštvo podataka, brzinu svakog automobila posebno, boju, proizvođača, model itd. Takođe bi vam bio potreban veliki broj funkcija da biste kontrolisali svaki automobil, kao što su upravljanje, uključivanje i isključivanje farova, ubrzavanje i kočenje. U OOP jeziku jednostavno radite sa kolekcijom automobil-objekata. Možete lako da stvorite više automobil-objekata koji se ponašaju kao nacrt (Blueprint). Nacrt sadrži sve podatke o automobilu i kompletan program potreban za manipulisanje automobilom. Kada napravite novi automobil-objekat, nacrt prenosi sve informacije do novog objekta.

Na primer, svaki automobil može imati sledeća *svojstva:*

• *boja,*

*• brzina,*

*• proizvođač,*

*• model.*

a pored toga, može imati iste *metode:*

• *ubrzavanje,*

*• kočenje,*

*• skretanje,*

*• uključivanje/isključivanje farova.*

Svojstva opisuju objekat; metodi čine da objekat nešto radi. Pošto je svaki automobil-objekat stvoren iz istog nacrta, trenutna informacija unutar svakog objekta je jedinstvena i nije povezana sa vrednostima bilo kojeg drugog automobil-objekta (izuzev ako želite da tako bude). To znači da, iako svi automobil objekti imaju ista svojstva, jedan automobil može da bude plav, drugi crven.

Pošto definišete nacrt o načinu funkcionisanja vašeg objekta, na osnovu njega veoma lako možete stvarati veliki broj automobil-objekata. Ovo umnogome podseća na masovnu proizvodnju istog artikla baziranu na dizajnu samo jednog proizvoda.

**9.2 KREIRANJE I UPOTREBA OBJEKATA**

Flashkoristi veliki broj ugrađenih objekata. Oni obuhvataju sve vrste objekata, počevši od najosnovnijeg među njima – objekta tipa Object, pa sve do objekta tipa XMLSocket koji obavlja prenos XML podataka preko TCP/IP Socket-a.

Kada želite da kreirate vlastiti objekat koristeći neki od ugrađenih objekata u Flash-u upotrebljavate ključnu reč *new.* Tako, na primer, ukoliko želite da kreirate novi niz (*Array*) baziran na objektu *Array*, treba samo da pokrenete panel Actions i da otkucate sledeće:

*mojObjekat = new Array();*

Dakle, nakon reči *new* odredili smo predefinisani objekat na kojem će biti baziran novi objekat. Ne samo što smo odredili naziv objekta po želji, već smo istovremeno pozvali funkciju koja će kreirati taj objekat.

Za primer ćemo uzeti objekat koji čuva spisak za kupovinu. Možemo zamisliti takav spisak kao niz, odnosno skup „redova" sa rupama u kojima se čuvaju promenljive.

Prvi korak je da kreirate niz, a zatim da ga popunite. Objekat *Array* ima već definisano svojstvo *length* koje sadrži broj elemenata niza.

Niz ćete kreirati jednostavnom akcijom:

*Lista = new Array();*

Tako kreiran niz je još uvek prazan pa sada prelazimo na akcije kojima ćemo popuniti niz:

*Lista[0] = "hleb";*

*Lista[1] = "mleko";*

*Lista[2] = "voce";*

Ovime smo kreirali tri elementa niza (prvi indeks u nizu je nula), i ako sada upišemo akciju za listanje elemenata niza, videćemo da li je sve u redu:

*trace (Lista.length);*

*for (i=0; i<Lista.length; i++) {*

*trace (Lista[i]);*

*}*

Šta se ovde dešava? Prvim redom mi ispisujemo dužinu niza, tj. broj elemenata niza. Zatim uvodimo brojač (*i*) koji uzima vrednosti od nule do najvećeg indeksa niza i ispisujemo element niza sa tim indeksom u svakom ciklusu.

Kao rezultat u *Output* panelu bi trebalo da piše:

*3*

*hleb*

*mleko*

*voce*

Kao što vidite, ovde smo upotrebili ugrađeno svojstvo objekta *Array* koje ima naziv *length* i koje prikazuje broj elemenata niza. Osim ugrađenih svojstava, ovaj objekat ima i ugrađen metod za dodavanje elemenata niza *push*.

Ukoliko dodamo još jednu liniju koda:

*Lista.push("deterdzent");*

zapravo ćemo dodati novi element na kraj tog niza. Ako sada izlistamo sadržaj niza, videćemo da se i ovaj element nalazi u nizu. Dakle, kompletan kod treba da izgleda ovako:

*Lista = new Array();*

*Lista[0] = "hleb";*

*Lista[1] = "mleko";*

*Lista[2] = "voce";*

*Lista.push("deterdzent");*

*trace ("Prva lista: " + Lista.length);*

*for (i=0; i<Lista.length; i++) {*

*trace (Lista[i]);*

*}*

U prvom delu koda kreiramo niz i njegove elemente, a u drugom delu samo listamo sadržaj niza. Poenta je da svi ugrađeni objekti u Flash-u imaju svoja svojstva i metode koje je potrebno naučiti kako biste efikasno koristili te objekte.

**9.3 SPISAK NEKIH UGRAĐENIH OBJEKATA SA SVOJSTVIMA I METODIMA**

**Objekat Array**

*Konstruktor:*

*new Array()*

*new Array(length)*

*new Array(element0, ... elementN)*

Dakle, mogu se odmah kreirati i elementi niza.

*Metode*:

* *mojNiz.pop()* – uklanja poslednji element iz niza i vraća njegovu vrednost,
* *mojNiz.reverse()* – menja raspored elemenata u nizu (prvi i poslednji menjaju mesto, drugi i pretposlednji itd.),
* *mojNiz.sort()* – sortiranje niza bez pravljenja kopije,
* *mojNiz.toString()* – vraća elemente niza u obliku String-a,
* *mojNiz.shift()* – uklanja prvi element iz niza i vraća njegovu vrednost,
* *mojNiz.unshift(vrednost0,...,vrednostN)* – dodaje elemente na početak niza i vraća novu dužinu,
* *mojNiz.slice(pocetak, kraj)* – vrši izvlačenje elemenata niza koji se nalaze između elementa niza sa rednim brojem *početak* do elementa niza sa rednim brojem *kraj*.

**Objekat Date**

*Konstruktor*:

*new Date()*

*new Date (godina, mesec, dan, cas, minut, sekunda, milisekunda)*

Kreira novi objekat Date (datum) koji čuva trenutni datum i vreme ili datum koji korisnik odredi.

*Metode*:

* *mojDatum.getDate()* – vraća redni broj dana u mesecu,
* *mojDatum.getDay()* – vraća redni broj dana u nedelji,
* *mojDatum.getFullYear()* – vraća godinu u četvorocifrenom formatu,
* *mojDatum.getHours()* – vraća čas,
* *mojDatum.getMinutes()* – vraća minute,
* *mojDatum.getMonth()* – vraća redni broj meseca u godini,
* *mojDatum.setDate (dan)* – postavlja redni broj dana u mesecu i vraća novo vreme u milisekundama,
* *mojDatum.setFullYear(godina, mesec, dan)* – postavlja godinu i vraća novo vreme u milisekundama. Ako su zadati i dodatni parametri mesec i dan, i oni se postavljaju.

**Objekat Key**

*Metode*:

* *Key.addListener(noviOsluskivac)* – registruje objekat za prijem obaveštenja o događajima onKeyDown i onKeyUp,
* *Key.getAscii()* – vraća ASCII kod poslednjeg pritisnutog ili otpuštenog tastera,
* *Key.getCode()* – vraća kod tastera koji je poslednji pritisnut,
* *Key.isDown(kodTastera)* – vraća true ako je otpušten taster čiji je kod zadat kao vrednost parametra.

*Sintaksa osluškivača:*

*noviOsluskivac = new Object();*

*Key.addListener(noviOsluskivac);*

*noviOsluskivac.onKeyDown = function() { .... }*

*noviOsluskivac.onKeyUp = function() { ... }*

Funkcije služe kako bi nosile akcije koje će se pokrenuti na pritisak odnosno puštanje tastera.

**Objekat MovieClip**

*Metode:*

* *mojMC.createEmptyMovieClip(nazivMC, dubina)* – kreira prazan MovieClip simbol sa Instance Name-om koji je definisan,
* *mojMC.duplicateMovieClip(noviNaziv, objekat)* – kreira novu kopiju određenog MovieClip simbola u toku prikazivanja filma. Kopirani simbol uvek počinje prikazivanjem od frejma 1,
* *mojMC.getBytesLoaded()* – vraća broj učitanih bajtova,
* *mojMC.getBytesTotal()* – vraća ukupan broj bajtova MovieClip simbola,
* *mojMC.gotoAndPlay(frejm)* – započinje prikazivanje filma od određenog frejma,
* *mojMC.hitTest(cilj)* – proverava da li je MovieClip simbol preklopljen sa oblašću koju predstavlja parametar *cilj,*
* *mojMC.startDrag(katanac, levo, gore, desno, dole)* – omogućava prevlačenje (kursorom) simbola na neku drugu lokaciju. Katanac kao parametar određuje da li će simbol biti vezan za centralnu poziciju miša, a ostali parametri su opcijoni i određuju granice so kojih može biti prevlačen simbol,
* *mojMC.stopDrag()* – prekida prevlačenje simbola.

Ovde su prikazani samo neki metodi i svojstva, a kompletne spiskove objekata sa metodama i svojstvima možete pronaći na sajtovima podrške Adobe Flash-u. Najbolje je krenuti od sajta *www.actionscript.org* koji predstavlja odličnu bazu za početnike u objektno orjentisanom programiranju.

# 10.PRIMERI U ACTIONSCRIPTU

**Primer 1: Skočica**

Kreirajte klip koji ima grafiku u obliku lopte. Kopiju klipa možete nazvati myClip, ali u ovom slučaju kod neće zavisiti od naziva klipa. Klipu pridružite sledeći kod:

*onClipEvent (enterFrame){*

*this.\_x +=5;*

*}*

Ovaj kod se izvršava jednom na svakom frejmu. On pomera frejm za jedan piksel na svakom frejmu. Rezultat je klip koji se polako kreće preko ekrana sve dok ne dostigne drugu ivicu. On ustvari nastavlja da se kreće i preko ove ivice.

Da bi klip nastavio da se kreće i preko desne ivice ekrana potrebno je da izvršimo nekoliko izmena u kodu. Horizontalna brzina klipa će biti smeštena u promenljivoj nazvanoj *speedx.* Izvršićete izmene u skripti tako da ona sada izgleda ovako:

*onClipEvent(load){*

*speedX=5;*

*}*

*onClipEvent(enterFrame){*

*this.\_x+=speedX;*

*}*

Ako sada pokrenete film on će se ponašati kao i u prethodnom primeri. Promenljiva *speedX* je podešena na 5, a ta vrednost se koristi za povećanje horizontalne pozicije klipa.

Sada je vreme da napravimo klip koji prelazi desnu ivicu ekrana. Najpre ćemo proveriti da li je horizontalna pozicija klipa na ili van desne ivice ekrana. Ako jeste *speedX* je onda zabrana tako da se lopta vraća istim putem nazad.

*onClipEvent(load){*

*speedX=5;*

*}*

*onClipEvent(enterFrame) {*

*this.\_x += speedX;*

*if (this.\_x>=550) {*

*speedX=-speedX;*

*}*

*}*

Sada se lopta kreće van desne ivice ekrana i vraća nazad prema levoj ivici.

Da bismo u to bili sigurni da prelazi preko ivicu ekrana, potrebno je da proverimo da li je horizontalna lokacija lopte manja od 0 i takođe u tom slučaju preokrenuti njen smer.

*onClipEvent(load){*

*speedX=5;*

*}*

*onClipEvent(enterFrame){*

*this.\_x+=speedX;*

*if (this.\_x>=550){*

*speedX=-speedX;*

*} else if (this.\_x<=0){*

*speedX=-speedX;*

*}*

*}*

Sledeći korak je kreiranje lopte i u vertikalnom pravcu. I u ovom slučaju nema nekih naročitih izmena. Ta izmena obuhvata sve ovo što smo dosad uradili, samo ovoga puta primenjeno i na horizontalni i na vertikalni smer:

*onClipEvent(load){*

*speedX=5;*

*speedY=5;*

*}*

*onClipEvent(EnterFrame){*

*this.\_x +=speedX;*

*this.\_y+=speedY;*

*if (this.\_x>=550){*

*speedX=-speedX;*

*} else if (this.\_x<=0) {*

*speedX=-speedX;*

*}*

*If (this.\_y>=400){*

*speedY=-speedY;*

*} else if (this.\_y<=0){*

*speedY==speedY;*

*}*

*}*

Kada nakon ovoga startujete film, lopta će prelaziti izvan ivica sva četiri zida. I nastavljaće da skakuće. Ovo je dobar primer jedne animacije koja je lako urađena zahvaljujući ActionScriptu, za raliku od animacija koja se ručno kreira frejm po frejm.

**Primer 2: Simulacija 3D skaliranja**



Na slici prikazan je isti klip na dvema različitim lokacijama na ekranu. Radi se o istom klipu, osim što je klip u donjem levom uglu više skaliran u odnosu na onaj u gornjem desnom uglu. To stvara privid da je prvi klip bliži. Ovaj privid se može i dalje povećavati zahvaljujući kretanju. Sada ćemo kreirati film na kojem će avion putovati iz jednog ugla ekrana u drugi, uz istovremene promene u skaliranju.

Kreirajte klip sa grafikom aviona recimo, ili nekog drugog objekta. Klip postavite bilo gde na ekranu i pridružite klipu sledeći kod:

*onClipEvent(load) {*

*scaleAmt = 10;*

*x = 525;*

*y = 25;*

*}*

*onClipEvent(enterFrame) {*

*scaleAmt++;*

*x -= 5;*

*y += 5;*

*this.\_xscale = scaleAmt;*

*this.\_yscale = scaleAmt;*

*this.\_x = x;*

*this.\_y = y;*

*}*

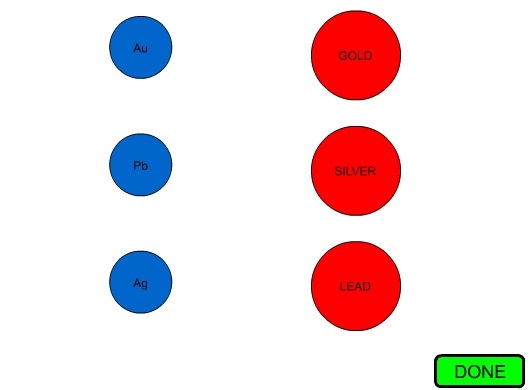
U prvom delu ove skripte podešavamo tri promenljive na njihove početne vrednosti. Promenljiva *scaleAmt* sadrži vrednost scale klipa. Promenljive *x* i *y* su lokacije klipa.

Nakon svakog frejma ove vrednosti će se menjati. Promenljiva *scaleAmt* se povećava za 1. Promenljiva *x* se smanjuje za 5 pomerajući klip ulevo. Promenljiva *y* se povećava za 5 što prouzrokuje spuštanje klipa. I *\_xscale*  i *\_yscale* atribudi klipa su podešeni na *ScaleAmt.*

Rezultat ovog filma je da će avion poleteti iz gornjeg desnog ugla prema donjem levom uglu, delujući kao da se priblizava korisniku. U ovom slučaju izgleda kao da se avion posmatra odozgo. Ukoliko nacrtamo avion u nomalnoj perspektivi, moguće ga je pikazati kao da se približava ili udaljuje u bilo kom pravcu.

**Primer 3: Igra sparivanja**

Kreiraćemo igru gde će korisnik morati da sparuje klipove na jednoj strani ekrana sa klipovima na drugoj. Tri klipa na levoj strani se mogu prevlačiti. Tri klipa na desnoj strani su drop zone i imaju nazive *dropZone1, dropZone2* i *dropZone3*. Svih 6 kopija klipova su kreirani od različitih klipova iz biblioteke. To je zato što je njihov sadržaj teksta raličit.



Možete kreirati film sa 6 različitih klipova kao na slici i button simbol (dugme) *Done.* Samo je potrebno da tri drop zone budu odgovarajuće nazvane.

Obratite pažnju da drop zone selektujete jednu po jednu, a zatim *Modify, Arrange, Send to back,* da bi bili sigurni da će one po redosledu biti iza klipova koji se mogu prevlačiti, kada ih korisnik premesti na isti prostor.

Obratite pažnju da klipovi nisu svrstani ni po kakvom redu. *Au* je zlato, *Ag* srebo i *Pb* je olovo. Tako da će korisnik morati da ih sredi. Ceo kod se nalazi u skriptama klipa pridruženim na klipove na levoj strani. Prvo, originalna pozicija klipa je zapisana i smeštena u promenljivim *origx* i *origy.* Zatim, ako se klip ne nalazi iznad odgovarajuće zone prevlačenja, \_*x* i *\_y* lokacije se resetuju na ove početne vrednosti. Međutim, ako se klip nalazi iznad odgovarajuće zone prevlačenja, lokacija klipa se može prevlačiti je podešena na centar zone prevlačenja. Sledi primer tog koda:

*onClipEvent (load) {*

*origX = this.\_x;*

*origY = this.\_y;*

*}*

*onClipEvent (mouseDown) {*

*if (this.hitTest(\_root.\_xmouse, \_root.\_ymouse)) {*

*this.startDrag();*

*}*

*}*

*onClipEvent (mouseUp) {*

*if (this.hitTest(\_root.\_xmouse, \_root.\_ymouse)) {*

*this.stopDrag();*

*// proverava da li se dropZone nalazi u centru svog movie clipa*

*if (\_parent.dropZone1.hitTest(this.\_x,this.\_y,true)) {*

*// nalazi se u centru*

*this.\_x = \_parent.dropZone1.\_x;*

*this.\_y = \_parent.dropZone1.\_y;*

*} else {*

*// vraća na originalnu lokaciju*

*this.\_x = origX;*

*this.\_y = origY;*

*}*

*}*

*}*

Ostali klipovi imaju isti kod, izuzev što je *dropZone1* zamenjen sa *dropZone2* i *drop Zone3* na sva tri mesta na kojima se pojavljuje u kodu.

Kod koji proverava da li je korisnik dobio prave odgovore nalazi se u button simbolu *Done:*

*on (release) {*

*// preuzima sve tačne*

*correct = true;*

*// proverava da li se drag mc lokacija poklapa*

*// drop zone lokacije*

*if ((drag1.\_x != dropZone1.\_x) or (drag1.\_y != dropZone1.\_y)) {*

*correct = false;*

*}*

*if ((drag2.\_x != dropZone2.\_x) or (drag2.\_y != dropZone2.\_y)) {*

*correct = false;*

*}*

*if ((drag3.\_x != dropZone3.\_x) or (drag3.\_y != dropZone3.\_y)) {*

*correct = false;*

*}*

*// rezultat na Output prozoru*

*if (correct) {*

*trace("Sve je tacno!");*

*} else {*

*trace("Nije kompletirano.");*

*}*

*}*

**Primer 4: Preloader**

Preloader je animacija koja se pojavljuje za vreme učitavanja Flash filma, bilo da je reč o sajtu, aplikaciji ili jednostavnoj animaciji. Preloader ima smisla samo ako Flash fajl postavljate na Internet, tj. na neki server jer samo tako postoji učitavanje. Ako je fajl u lokalu, tj. na računaru korisnika, on će se otvoriti u istom trenutku, tako da nije potrebno praviti sekvencu učitavanja.

Za kreiranje Preloader-a biće nam potreban Flash fajl koji ima određenu veličinu (ako je veličina fajla isuviše mala, ne treba nam Preloader). Takođe, treba izdvojiti prostor u Timeline-u za Preloader, svakako na početku filma (tj. pre samog sadržaja). Najbolje je izdvojiti posebnu scenu za to, a uz ovu lekciju dolazi i fajl za vežbanje.

Preloader se sastoji iz nekoliko elemenata:

1. Tekstualni okvir koji u Variable polju ima upisano *per*.
2. Pravougaonik koji će se širiti tako da prikazuje progres učitavanja sajta (zato se i naziva Progress bar i ima Instance Name *progress\_bar*).

Actionscript kod koji ćemo dodati moramo postaviti na neki MovieClip simbol jer je potrebno da se taj kod izvršava u svakom frejmu Preloading animacije (25-35 puta u sekundi kako bi često računao procenat učitanog sajta). Sam kod koji ćemo dodati bilo kom MovieClip simbolu na sceni izgleda ovako (pošto imamo samo jedan simbol, pravougaonik koji predstavlja Progress bar, dodaćemo ga njemu):

*onClipEvent (enterFrame) {*

*\_parent.loading = \_parent.getBytesLoaded();*

*\_parent.total = \_parent.getBytesTotal();*

*\_parent.percent = (\_parent.loading/\_parent.total)\*100;*

*\_parent.per=int(\_parent.percent);*

*\_parent.progress\_bar.\_width = \_parent.per\*8;*

*if (\_parent.per>99) {*

*\_parent.play();*

*}*

*}*

Analizom koda možemo najpre videti da se zbog prvog reda poziva funkcija u svakom pojavljivanju simbola u frejmu (dakle, 25-35 puta u sekundi, zavisno od Frame Rate-a). Dalje se kreiraju nove promenljive *loading* i *total* čijim deljenjem dolazimo do promenljive *percent* koja predstavlja procenat učitanog sajta. Zaokruživanjem te promenljive kreiramo promenljivu *per* koja će se pojaviti u Dynamic text field-u. Širinu MovieClip simbola *progress\_bar* kontrolišemo takođe u ovom kodu i dodeljujemo mu vrednost promenljive *per* pomnoženu sa 8 kako bi se po učitanom sajtu taj simbol „razvukao“ po celoj širini filma od 800 px. Taj množilac možete menjati u zavisnosti od potrebe.

Na kraju, imamo uslov koji se ispituje u svakom frejmu, te ako je procenat učitanosti sajta preko 99% Flash pušta dalje film, tj. kreće da prikazuje sadržaj.

Pored ovog koda, potrebno je u prvom frejmu dodati i akciju:

*stop();*

kako bi Flash zaustavio prikaz filma sve dok vrednost promenljive *per* ne pređe 99%.

Sada smo kreirali Preloader koji ćemo testirati tako što ćemo najpre testirati film pritiskom na CRTL+Enter, a kada se otvori test prozor, još jednom pritisnuti CTRL+Enter za simulaciju Download-a.

Za vežbu kreirajte novi fajl, na jednoj sceni postavite neki sadržaj koji će povećati veličinu izlaznog fajla (video ili jpeg sliku velikih dimenzija), a zatim na novoj sceni (koja će u spisku scena biti ispred glavne scene) napravite Dynamic Text Field (koji će imati vrednost promenljive *per*) i pravougaonik koji će biti MovieClip simbol sa Instance Name-om *progress\_bar*, a zatim dodajte Preloader kod tom simbolu i testirajte film.

**Primer 5: Tekst koji prati kursor**

Ovaj kod možete koristiti kao zgodan efekat pri učitavanju sajta. Dok se sajt učitava, pojavljuje se tekst koji prati kursor dok se u samom Preloader-u daje prikaz procenta učitanog sajta.

Implementacija koda je vrlo jednostavna i potrebno je samo postaviti sledeći kod u ključni frejm u kojem želite da se pojavi tekst:

*Text = "Tekst koji prati kursor";*

*letters = Text.split("");*

*letterformat = new TextFormat();*

*letterformat.font = "Verdana";*

*letterformat.align = "center";*

*letterformat.size = "10";*

*spacing = 8;*

*speed = 3;*

*for (var LTR = 0; LTR<letters.length; LTR++) {*

*mc = \_root.createEmptyMovieClip(LTR+"l", LTR);*

*mc.createTextField(letters[LTR]+"t", LTR, LTR\*spacing, 10, 20,*

*20);*

*with (mc[letters[LTR]+"t"]) {*

*text = letters[LTR];*

*setTextFormat(letterformat);*

*selectable = false;*

*} if (LTR) {*

*mc.prevClip = \_root[(LTR-1)+"l"];*

*mc.onEnterFrame = function() {*

*this.\_x += (this.prevClip.\_x-this.\_x+5)/speed;*

*this.\_y += (this.prevClip.\_y-this.\_y)/speed;*

*}*

*} else {*

*mc.onEnterFrame = function() {*

*this.\_x += (\_root.\_xmouse-this.\_x+10)/speed;*

*this.\_y += (\_root.\_ymouse-this.\_y)/speed;*

*}*

*}*

*}*

Promenljiva *Text* sadrži tekst koji treba da se pojavljuje pored kursora, a parametar *speed* nosi informaciju o brzini praćenja (što je manja vrednost, tekst se brže pozicionira).

Promenljiva *letters* predstavlja niz slova, a u sledećim redovima koda se vrši formatiranj slova (tip fonta, veličina, centriranje).

Sam kod najpre razdvaja slova teksta, a zatim kreira MovieClip koji sadrži pojedinačna slova.

Kod se izvršava u svakom frejmu (onEnterFrame) i u svakom frejmu se slovima pomera pozicija kako bi ispratila kursor miša.

# ZAKLJUČAK

Činjenica da predstavljeni primeri imaju veličinu korisničkog fajla (.swf filma) od par desetina KB, a da u potpunosti definišu ponašanje aplikacije i realizuju kompletan korisnički interfejs, nedvosmisleno govori u prilog tvrdnji da Flash filmovi jesu pravo rešenje u svetu Interneta.

Vektorska grafika i animacija Flasha i jezik ActionScript svojom raznovrsnošću omogućavaju rešavanje širokog spektra web problematike. Od kreiranja navigacionih rešenja u okviru HTML stranica, animiranih reklamnih sličica, do izrade kompletnih web-prezentacija u Flashu i direktnu razmenu podataka sa web serverom i njihovu dalju obradu.

# LITERATURA

*[1] Flash Mx ActionScript –* Autor: Gary Rosenzweing

*[2] Flash ActionScript –* Autor: Jobe Makar

*[3] www.actionscript.org*

*[4] http://www.link-elearning.com –* Kurs ActionScipt-a

*[5] http://www.developphp.com/list\_as3.php -* Learn ActionScript 3.0