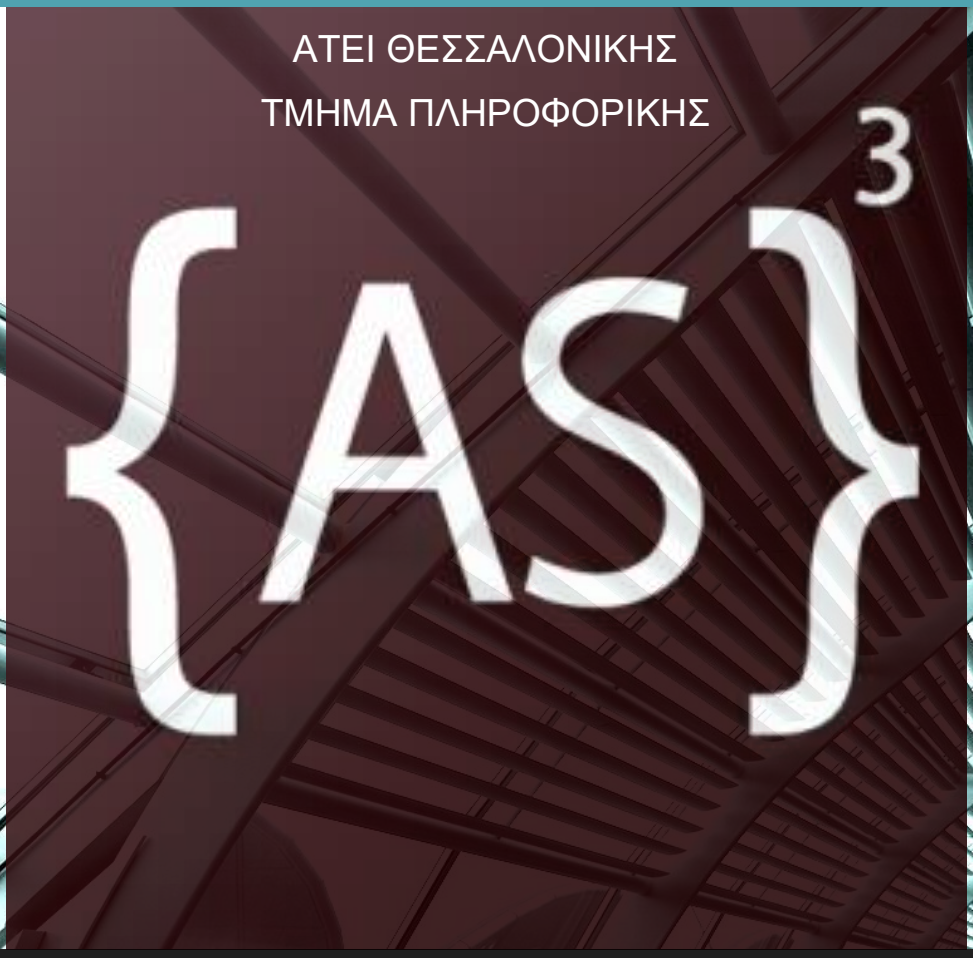
ΑΤΕΙ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή Εργασία

Ανάπτυξη Λογισμικού για την Δημιουργία Εκπαιδευτικών Εργαλείων σχετικών με την Οργάνωση και Αρχιτεκτονική Υπολογιστών

Του φοιτητή
Κωνσταντίνου Κοντογιώργου
Αρ. Μητρώου: 2078

Επιβλέπων καθηγητής
Βαφειάδης Αντώνης

ΙΟΥΝΙΟΣ 2010

ΕΥΡΕΤΗΡΙΟ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΡΟΛΟΓΟΣ

Η ένταξη της Πληροφορικής ως βασικού εκπαιδευτικού εργαλείου και η παράλληλη εκμετάλλευση του υπάρχοντος επιστημονικοτεχνικού δυναμικού έχει αποτελέσει σημαντικό βήμα για την ενσωμάτωση και την εφαρμογή της στην εκπαιδευτική διαδικασία. Τα τελευταία χρόνια αναπτύσσεται σημαντικά ο ρόλος της Πληροφορικής και των Νέων Τεχνολογιών (πολυμέσα, εικονική πραγματικότητα, Διαδίκτυο κ.λ.π.) με εφαρμογές που προέρχονται από ερευνητικά-ακαδημαϊκά ιδρύματα και από ιδιωτικούς φορείς.

Οι σύγχρονες εφαρμογές εκπαιδευτικού λογισμικού βασίζονται στην τεχνολογία των υπερμέσων. Τα κύρια χαρακτηριστικά των υπερμέσων. που μπορούν να αξιοποιηθούν στην εκπαιδευτική διαδικασία, είναι:

- η δυνατότητα χρήσης πολλαπλών αναπαραστάσεων για την παρουσίαση φαινομένων και καταστάσεων
- η μη γραμμική οργάνωση και προσπέλαση των πληροφοριών
- η μεταβολή των μαθησιακών δραστηριοτήτων, μέσω της αναζήτησης των εννοιολογικών δομών και συσχετίσεων μεταξύ των διαφόρων πληροφοριών από τους ίδιους τους μαθητές
- η ανάπτυξη του κινήτρου μάθησης.

ΠΕΡΙΛΗΨΗ

Η δημιουργία του περιβάλλοντος διεπαφής και η σύνθεση των πολυμεσικών στοιχείων έγινε χρησιμοποιώντας το λογισμικό Adobe Flash CS4. Το Flash είναι ένα από τα πιο διαδεδομένα εργαλεία ανάπτυξης πολυμεσικών παρουσιάσεων και εφαρμογών επαγγελματικού επιπέδου. Είναι ένα πλήρες σύστημα απο ολοκληρωμένα tools, frameworks, clients και servers τα οποία μπορούν να χρησιμοποιηθούν για τη δημιουργία και ανάπτυξης εφαρμογών Web, περιεχόμενα και video που «τρέχουν» σε διάφορα λειτουργικά συστήματα και συσκευές μέσω του Adobe Flash Player και Adobe AIR.

Διαθέτει ενσωματωμένη μία ισχυρή γλώσσα προγραμματισμού script την Actionscript. Η πιο πρόσφατη έκδοση της Actionscript είναι η AS3. Χρησιμοποιείται για τον προγραμματισμό interactive περιεχομένου για την πλατφόρμα Adobe Flash. Η ActionScript 3 δημιουργήθηκε προκειμένου να παρέχει σημαντικές βελτιώσεις στην επίδοση, σε σχέση με την Actionscript 2, και να κάνει την ανάπτυξη των εφαρμογών και τη συντήρηση ευκολότερη καθώς επίσης προσανατολίζεται προς τον αντικειμενοστραφή προγραμματισμό.

ABSTRACT

The creation of the interface and the composition of multimedia data were performed using the software Adobe Flash CS4. Flash is one of the most commonly used tools for developing multimedia presentations and professional applications. It is a complete system of integrated tools, frameworks, clients and servers that can be used for the creation and development for web applications, and video content to "run" with the result in different operating systems and devices using the Adobe Flash Player and Adobe AIR.

It has built a strong programming language script to Actionscript. The latest version of Actionscript is AS3. It is used for programming interactive content platform for Adobe Flash. The Actionscript 3 was created to provide significant improvements in performance, compared with Actionscript 2, and make applications development and maintenance easier and also the object-oriented programming.

ΕΙΣΑΓΩΓΗ

Αυτή η εργασία αποτελεί μια εισαγωγή στην ActionScript. και αναφέρεται στις βασικές διαφορές μεταξύ ActionScript 2 και ActionScript 3. Απευθύνεται στους σχεδιαστές και developers που έχουν ήδη κάποια εμπειρία με το περιεχόμενο των scripts που βρίσκονται στο Adobe Flash Professional.

Η εργασία αυτή μπορεί να χρησιμοποιηθεί ως μία εισαγωγή στην ActionScript 3 ή ως μια αναφορά σχετικά με συχνά θέματα που αφορούν τη δημιουργία ενός περιεχομένου σε Flash.

Αναπτύσονται εφαρμογές όπου προσομοιώνουν θέματα που αφορούν την οργάνωση και τις αρχιτεκτονικές των ηλεκτρονικών υπολογιστών με απώτερο σκοπό την καλύτερη κατανόηση των παραπάνω.

ΚΕΦΑΛΑΙΟ 1 – Εισαγωγή στην ActionScript

1.1 Τι είναι η ActionScript

Η ActionScript είναι μία script γλώσσα βασισμένη στην ECMAScript. Χρησιμοποιείται κυρίως για την ανάπτυξη ιστοσελιδών ή λογισμικού χρησιμοποιώντας την πλατφόρμα Adobe Flash Player (σε μορφή αρχείων SWF), αλλά επίσης χρησιμοποιείται σε μερικές εφαρμογές βάσεων δεδομένων (όπως η Alpha Five) και στην ρομποτική (όπως το Make Controller Kit). Αρχικά, αναπτύχθηκε από την Macromedia, αλλά η γλώσσα τώρα ανήκει στην Adobe (η οποία αγόρασε την Macromedia το 2005). Η ActionScript αρχικά σχεδιάστηκε για τον έλεγχο απλών δισδιάστατων (2D) ανυσματικών animation αναπτύσοντάς τα με το Adobe Flash (τότε γνωστό ως Macromedia Flash). Σε επόμενες εκδόσεις προστέθηκαν λειτουργίες οι οποίες επέτρεπαν τη δυνατότητα δημιουργίας διαδικτυακών παιχνιδιών και εφαρμογών RIA (Rich Internet Applications) καθώς και streaming media (όπως βίντεο ή ήχος).

1.2 Ιστορική αναδρομή

Η ActionScript ξεκίνησε ως μία γλώσσα script για το Macromedia Flash (που τώρα αναπτύσεται από την Adobe Systems ως Adobe Flash). Οι πτώτες τρεις εκδόσεις του Flash παρείχαν περιορισμένα χαρακτηριστικά διαδραστικότητας. Οι πρώτοι Flash developers μπορούσαν να αναθέσουν μια απλή εντολή, αποκαλώντας την “action”, σε ένα button ή frame. Το σύνολο των actions ήταν βασικές εντολές πλοήγησης όπως: “play”, “stop”, “getURL” και “gotoAndPlay”.

Με την έκδοση του Flash 4 το 1999, αυτό το από set με actions μετατράπηκε σε μια μικρή γλώσσα script. Νέες δυνατότητες εισήχθησαν για το Flash 4 περιλαμβάνοντας μεταβλητές (variables), εκφράσεις (expressions), operators, συνθήκες if και βρόγχοι (loops). Παρ’ όλο που εκ των έσω αναφερόταν ως “ActionScript”, το εγχειρίδιο χρήστη του Flash 4 και όλα τα εμπορικά έγγραφα συνέχισαν να χρησιμοποιούν τον όρο “actions” για να περιγράψουν αυτό το set εντολών.

1.3 Χρονοδιάγραμμα των flash players

- **Flash Player 2**: Η πρώτη έκδοση με υποστήριξη scripting. Οι εντολές (actions) που περιλαμβάνονται είναι οι gotoAndPlay, gotoAndStop, nextFrame και nextScene για τον έλεγχο του timeline.
- **Flash Player 3**: Επέκταση της βασικής υποστήριξης scripting support με τη δυνατότητα να φορτώνονται εξωτερικά SWF. (loadMovie).
- **Flash Player 4**: Ο πρώτος player με την πραγματοποίηση πλήρων εντολών scripting (*Actions*). Περιλαμβάνει την υποστήριξη των loops, συνθήκες, μεταβλητές και άλλες βασικές δομές μιας γλώσσας.
- **Flash Player 5**: Περιλαμβάνει την πρώτη έκδοση της ActionScript. Χρησιμοποιεί prototype-based προγραμματισμό βασισμένο στην ECMAScript, και την δυνατότητα αντικειμενοστραφούς προγραμματισμού.
- **Flash Player 6**: Προστέθηκε ένα μοντέλο χειρισμού γεγονότων (event handling model), υποστήριξη της συνθήκης switch. Η πρώτη έκδοση που υποστηρίζει τα πρωτόκολα AMF και RTMP τα οποία πραγματοποιούν το audio/video streaming.
- **Flash Player 7**: Προστίθενται επιπλέον χαρακτηριστικά όπως: CSS styling για μορφοποίηση κειμένου και υποστήριξη της ActionScript 2.0, μία γλώσσα προγραμματισμού βασισμένη στην ECMAScript 4 Netscape Proposal. **Flash Player 8**: Οι ActionScript 1 και ActionScript 2 επεκτείνονται με την προσθήκη νέων βιβλιοθηκών (libraries) με APIs για τον έλεγχο των γραφικών bitmap, ανέβασαμα αρχείων (file uploads) και live φίλτρα για blur και dropshadow.
- **Flash Player 9 (αρχικά ονομάζεται 8.5)**: Προσθτίθεται η ActionScript 3.0 με την έλευση ενός νέου virtual machine, το AVM2 (ActionScript Virtual Machine 2), το οποίο συνυπάρχει με το προηγούμενο (AVM1). Η απόδωση βελτιώνεται εφόσον ένας βασικός στόχος της έκδοσης του player περιλαμβάνει έναν νεό compiler. Προστίθεται και η υποστηρίξη XML, full-screen mode και Regular expressions. Αυτή είναι η πρώτη έκδοση του player με τον τίτλο Adobe Flash Player.

- **Flash Player 10 (αρχικά ονομάζεται Astro)**: Προσθήκη βασικών 3D χειρισμών, όπως η περιστροφή στον άξονα X, Y, και Z.. Διάφορες διεργασίες που αφορούν την επεξεργασία γραφικών ανατίθενται στην GPU με αποτέλεσμα την σημαντική μείωση του χρόνου για το rendering, με απόδοση σε υψηλότερα frame rates, ιδιαίτερα με το H.264 video. Επιπρόσθετα το Flash Player 10 υποστηρίζει επικοινωνία Peer to Peer (P2P) με το Real Time Media Flow Protocol (RTMFP).

1.4 Χρονοδιάγραμμα ανά έκδοση ActionScript

2000–2003: ActionScript "1.0" Με την έκδοση του Flash 5 τον Σεπτέμβριο του 2000, τα "actions" του Flash 4 βελτιώθηκαν περισσότερο και ονομάστηκαν "ActionScript" για πρώτη φορά. Αυτή ήταν η πρώτη έκδοση της ActionScript με επιρροές από την JavaScript και το πρότυπο ECMA-262 (3η έκδοση. Οι τοπικές μεταβλητές έπρεπε να δηλώνονται με την δήλωση var και μπορούσαν να δημιουργηθούν συναρτήσεις (functions) με έλεγχο παραμέτρων και τιμές επιστροφής. Το αξιοσημείωτο είναι πως η ActionScript μπορούσει τώρα να πληκτρολογηθεί σε έναν επεξεργαστή κειμένου παρά να συνταχθεί επιλέγοντας actions από drop-down lists και dialog boxes.

Με την επόμενη έκδοση του εργαλείου σύνταξης (authoring tool), το Flash MX, και τον αντίστοιχο Player (Flash Player 6), δεν έγιναν σημαντικές αλλαγές στη γλώσσα. Υπήρξαν μόνο μερικές μικρές αλλαγές, όπως η προσθήκη της συνθήκης switch και του τελεστή (===), οι οποίες έφεραν τη γλώσσα ποιο κοντά στο πρότυπο ECMA-262. Δύο σημαντικά χαρακτηριστικά που ξεχωρίζουν την ActionScript από προηγούμενες εκδόσεις είναι το loose type system και η εξάρτηση στην κληρονομικότητα prototype-based. Ο όρος loose typing αναφέρεται στην δυνατότητα μιας μεταβλητής να «κρατήσει» οποιονδήποτε τύπο δεδομένων. Αυτό ανοίγει την δυνατότητα για ποιο γρήγορη αναπτυξη σεναρίων (scripts) και είναι κυρίως ιδανικό για μικρού μεγέθους scripting projects. Ο όρος Prototype-based inheritance αποδίδεται στον μηχανισμό της ActionScript 1.0 για επαναχρησιμοποίση κώδικα και αντικειμενοστραφούς προγραμματισμού. Εκτός από την λέξη κλειδί μιας κλάσης που καθορίζει τα κοινά της χαρακτηριστικά, η

ActionScript 1.0 χρησιμοποιεί ένα ειδικό αντικείμενο το οποίο χρησιμοποιείται ως πρώτυπο (prototype) για μια κλάση αντικειμένων. Όλα τα κοινά χαρακτηριστικά μια κλάσης ορίζονται στο αντικείμενο πρότυπο της κλάσης και κάθε στιγμιότυπο (instance) περιέχει ένα σύνδεσμο (link) σε εκείνο το αντικείμενο πρότυπο.

2003–2006: ActionScript 2.0 Η επόμενη αναθεωρημένη έκδοση της γλώσσας είναι η ActionScript 2.0, η οποία παρουσιάστηκε τον Σεπτέμβριο του 2003 με την έκδοση του Flash MX 2004 και τον αντίστοιχο player, τον Flash Player 7. Ως μια ανταπόκριση των απαιτήσεων των χρηστών για μια γλώσσα καλύτερα εφοδιασμένη για μεγαλύτερες και πιο πολύπλοκες εφαρμογές, η ActionScript 2.0 περιλαμβάνει τα χαρακτηριστικά: compile-time, type checking και class-based σύνταξη, με τις λέξεις κλειδιά class και extends. Με την ActionScript 2.0 οι developers μπορούν να περιορίσουν τις μεταβλητές σε ένα συγκεκριμένο τύπο δεδομένων με το να προσθέσουν μια επισήμανση του τύπου έτσι ώστε, την ώρα του compile, να μπορέσουν να εντοπιστούν πιθανά λάθη σε σχέση με τον συνταιρισμό των τύπων των μεταβλητών. Η ActionScript 2.0 επίσης παρουσίασε τη σύνταξη σε σχέση με την κληρονομικότητα των κλάσεων, έτσι ώστε οι developers να μπορούν να δημιουργούν κλάσεις και interfaces, με τον ίδιο τρόπο με τον οποίο δημιουργούν σε άλλες γλώσσες βασισμένες σε κλάσεις όπως η Java και η C++. Αυτή η έκδοση συμβαδίζει μερικώς με τις προδιαγραφές της τέταρτης έκδοσης του προτύπου ECMAScript.

2006–σήμερα: ActionScript 3.0 Τον Ιούνιο του 2006, η ActionScript 3.0 εμφανίζεται για πρώτη φορά με το Adobe Flex 2.0 και τον αντίστοιχο player, τον Flash Player 9. Η ActionScript 3.0 αποτέλεσε μια θεμελιώδη αναδιάρθρωση της γλώσσας, τόσο πολύ ώστε χρησιμοποιήθηκε ένα πλήρως διαφοροποιημένο virtual machine. Το Flash Player 9 περιέχει δύο virtual machines, το AVM1 για κώδικα γραμμένο σε ActionScript 1.0 και 2.0, και το AVM2 για περιεχόμενο γραμμένο σε ActionScript 3.0. Στην Actionscript 3.0 προστέθηκε περιορισμένη υποστήριξη για hardware acceleration (DirectX, OpenGL).

ΚΕΦΑΛΑΙΟ 2 – Παραδείγματα σε ActionScript 2.0

Σε αυτό το κεφάλαιο θα αναφερθούν βασικά στοιχεία που αφορούν την σύνταξη σε ActionScript 2 και θα υλοποιηθούν σχετικά παραδείγματα για την πλήρη κατανόηση.

2.1 Βασικός κύκλος απόκτησης – εκτέλεσης εντολών

Με το παράδειγμα του βασικού **κύκλου απόκτησης – εκτέλεσης εντολών (fetch – execution cycle)** θα επισημανθούν και θα αναπτυχθούν τα παρακάτω χαρακτηριστικά:

- Σύμβολα (symbols) και στιγμιοτύπα (instances)
- Dynamic text
- Actions
- Motion Tween
- Shape Tween
- Βασικές εντολές πλοήγησης (navigation)
- Built-in components

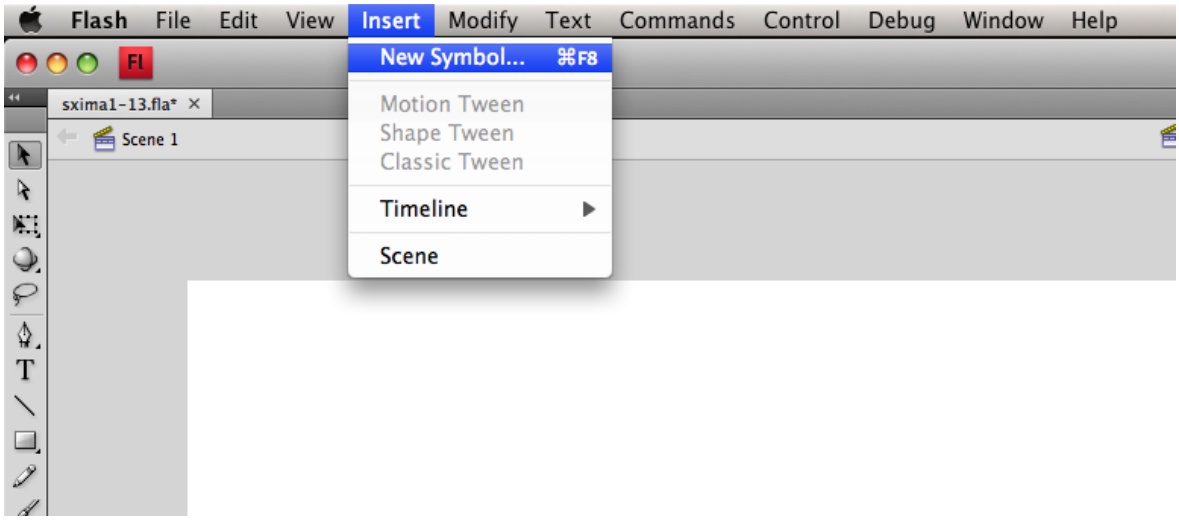
2.1.1 Χρήση συμβόλων και στιγμιοτύπων

Στο περιβάλλον Flash CS4 (και σε παλαιότερες εκδόσεις) παρέχεται η δυνατότητα δημιουργίας **συμβόλων (symbols).** Ένα σύμβολο δημιουργείται μία φορά μέσω των κλάσεων Button (ActionScript 2.0) ή SimpleButton (ActionScript 3.0) και MovieClip ή απ’ευθείας μέσα από το Flash authoring tool και γίνεται μέρος της βιβλιοθήκης (library). **Στιγμιότυπο (instance)** είναι ένα αντίγραφο ενός συμβόλου το οποίο μπορεί να τοποθετηθεί στο Stage ή να συγχωνευθεί σε άλλο σύμβολο. Ένα στιγμιότυπο μπορεί να είναι διαφορετικό από το σύμβολο που προήλθε σε χρώμα, μέγεθος ή λειτουργίες. Κάθε αλλαγή σε κάποιο σύμβολο επηρεάζει όλα τα στιγμιότυπά του, αλλά κάθε αλλαγή σε στιγμιότυπο επηρεάζει μόνο το ίδιο το στιγμιότυπο.

Το πλεονέκτημα που προκύπτει από τη χρήση των συμβόλων και των στιγμιοτύπων είναι πως μειώνεται δραματικά το μέγεθος του αρχείου flash. Αποθηκεύοντας διαφορετικά στιγμιότυπα ενός συμβόλου απαιτείται λιγότερος χώρος αποθήκευσης από το να αποθηκεύονταν πολλά αντίγραφα των περιεχομένων ενός συμβόλου. Για παράδειγμα, μπορεί να μειωθεί το μέγεθος του αρχείου flash μετατρέποντας static γραφικά, όπως μια φωτογραφία για background, σε σύμβολο και να επαναχρησιμοποιηθούν. Η χρήση συμβόλων μπορεί επίσης να επιταχύνει το playback του αρχείου SWF, επειδή ένα σύμβολο χρειάζεται να φορτωθεί στον Flash Player μόνο μία φορά.

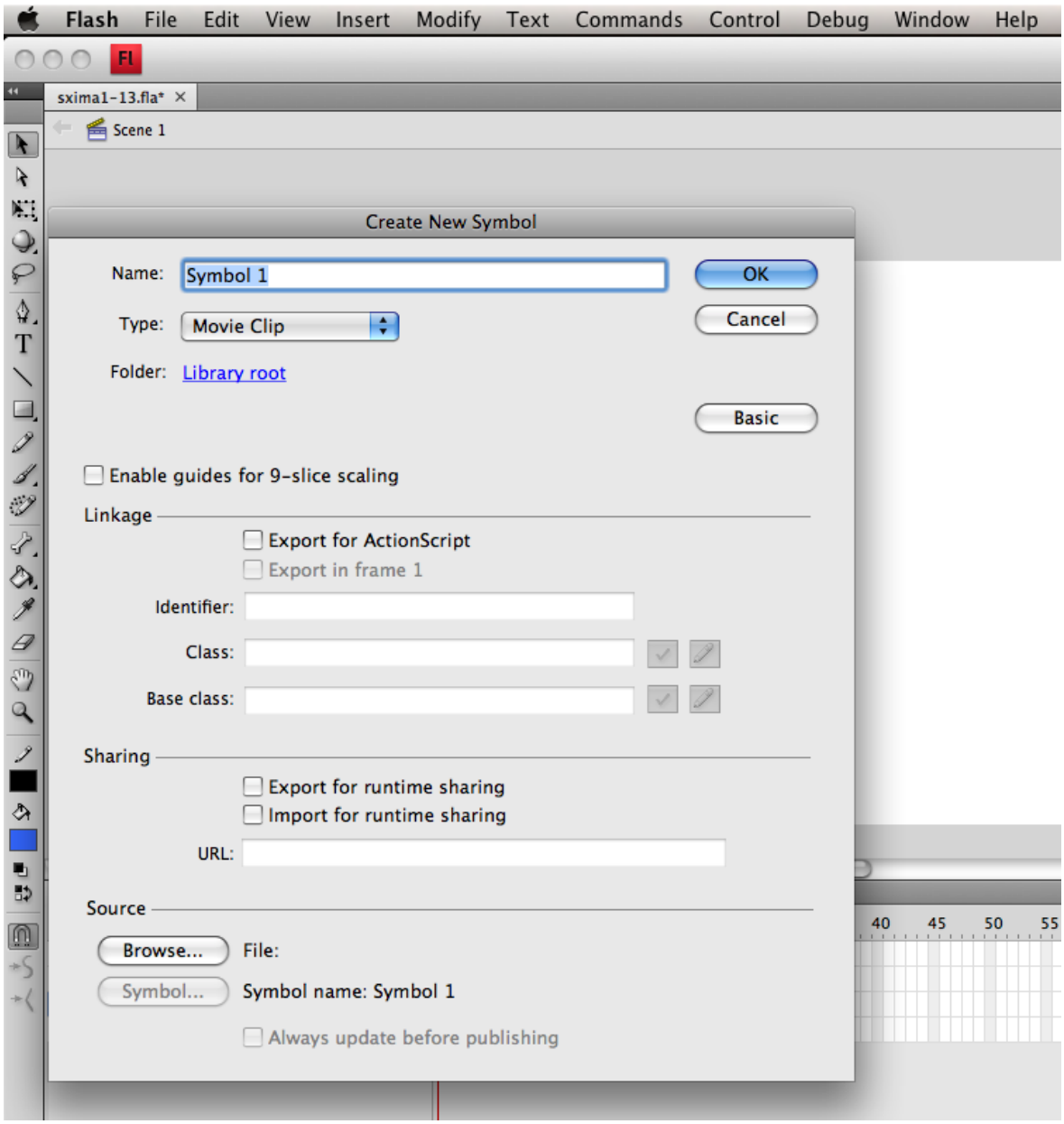
Δημιουργία κενού συμβόλου

1. Επιλέγουμε Insert -> New Symbol (Σχήμα 2.1.1)



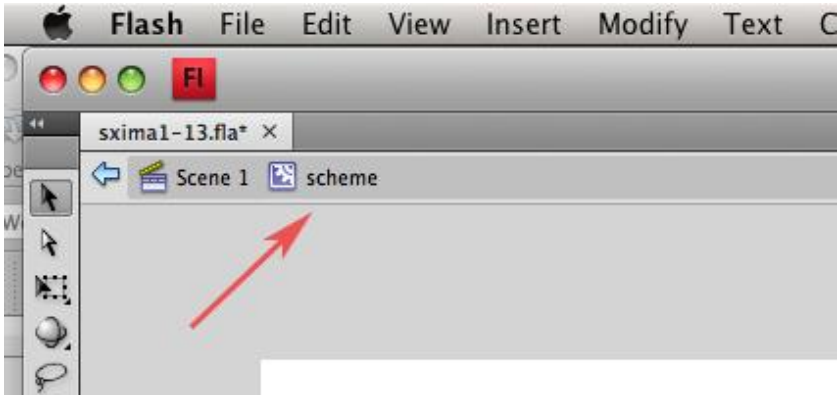
Σχήμα 2.1.1: Δημιουργία συμβόλου

2. Στο dialog-box, Πληκτρολογούμε το όνομα του συμβόλου και δηλώνουμε τον τύπο του (Movie Clip, Button, Graphic). (Σχήμα. 2.1.2)



Σχήμα 2.1.2: Ιδιότητες νέου συμβόλου

Για το παράδειγμα ακολουθούνται τα παραπάνω βήματα και δημιουργείται ένα σύμβολο με όνομα scheme και τύπο Movie Clip. Αυτό, θα αποτελέσει το βασικό Movie Clip όπου θα περιέχει τα επιμέρους σχήματα, σύμβολα και κείμενα που θα χρειαστούν για την υλοποίηση του παραδείγματος. Κάνοντας διπλό κλικ στο σύμβολο scheme που βρίσκεται στην βιβλιοθήκη παρατηρούμε πως το timeline, τα layers και τα frames ανήκουν όχι στο Stage αλλά στο συγκεκριμένο συμβόλο. (Σχήμα. 2.1.3)



Σχήμα 2.1.3: Το περιβάλλον ενός συμβόλου

2.1.2 Η εντολή stop();

Αρχικά δημιουργείται ένα layer με το όνομα Actions το οποίο θα περιλαμβάνει τα σχετικά Actions που αφορούν την πλοήγηση **(navigation controls).** Στο πρώτο keyframe πληκτρολογούμε την εντολή stop(); Όπου το αποτέλεσμα θα είναι να σταματήσει το playback στο 1^{ο} frame.

2.1.3 Δημιουργία στατικών και δυναμικών κειμένων

Με το Text Tool  δημιουργούμε στατικά κείμενα **(static text)** – σε ξεχωριστά layers- τα οποία περιλαμβάνουν τα κείμενα του παραδείγματος. Επίσης κατασκευάζεται ένα δυναμικό κείμενο **(dynamic text)** το οποίο θα εμφανίζει τον αριθμό του κύκλου απόκτησης – εκτέλεσης, λαμβάνοντας την τιμή από μία μεταβλητή που θα αυξάνεται με βήμα ένα, κάθε φορά που επαναλαμβάνεται ένας κύκλος.

2.1.4 Χρήση των built-in components

Στη συνέχεια, γίνεται χρήση των **built-in components**. Τα Adobe Flash Professional User Interface components είναι αντικείμενα κλάσεων της

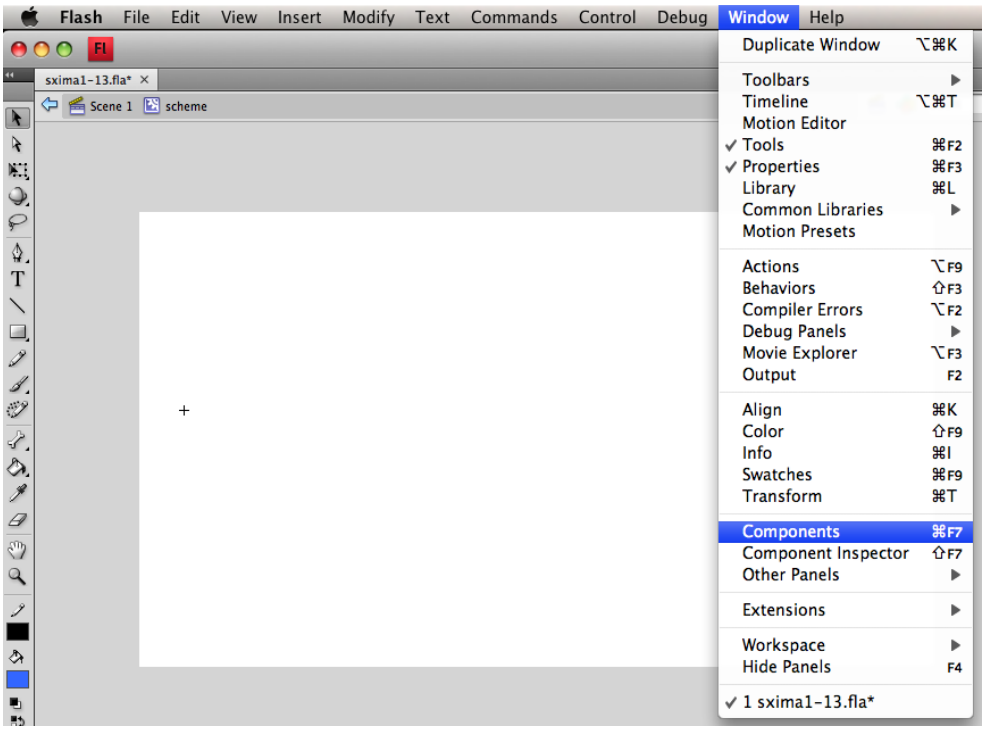
ActionScript 2.0 ή 3.0 τα οποία αποτελούν συχνά χρημοποιούμενα τμήματα του user interface μιας ιστοσελίδας ή εφαρμογής. Αυτά τα components βοηθούν στην επιτάχυνση της διαδικασίας ανάπτυξης μιας ιστοσελίδας ή εφαρμογής παρέχοντας στοιχειώδη τμήματα τα οποία μπορούν έυκολα να ενσωματοθούν και να τροποποιηθούν. Το Flash παρέχει τα παρακάτω components:

- Button.
- CheckBox
- ColorPicker
- ComboBox
- DataGrid
- Label
- List
- NumericStepper
- ProgressBar
- RadioButton
- ScrollPane.
- Slider
- TextArea
- TextInput
- TileList
- UILoader
- UIScrollBar

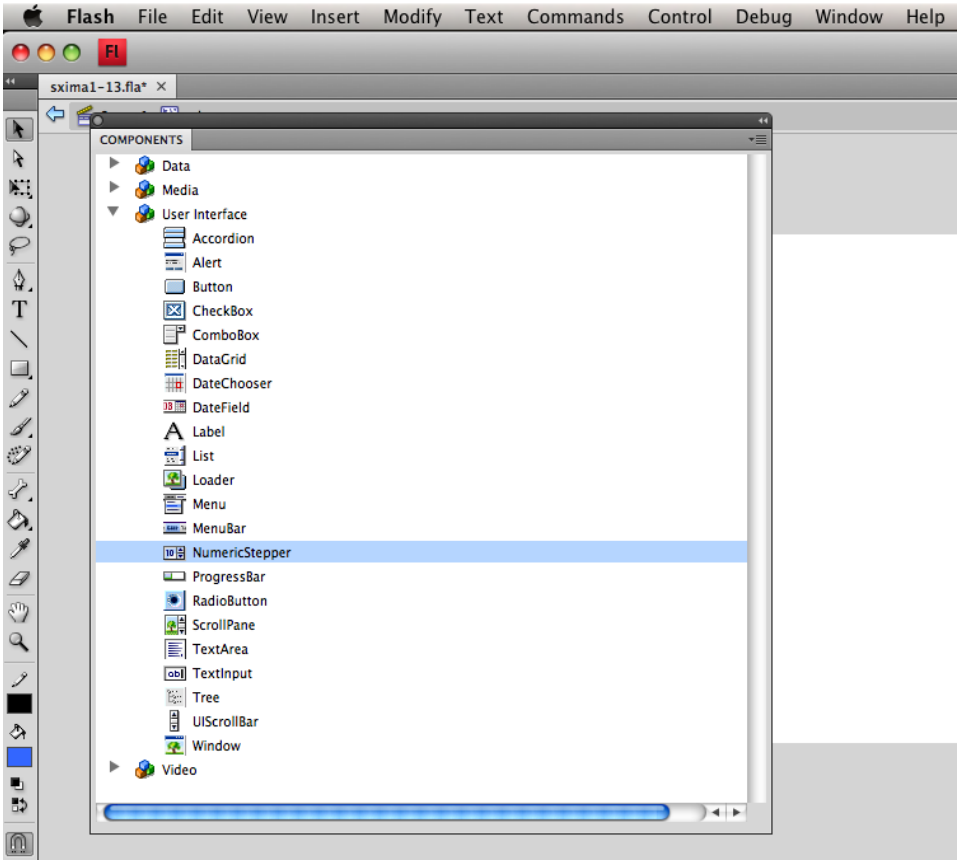
Στο παράδειγμα, γίνεται χρήση ενός NumericStepper με το οποίο ο χρήστης έχει τη δυνατότητα να ορίζει τον αριθμό των κύκλων που επιθυμεί να εκτελεστεί ο κύκλος απόκτησης – εκτέλεσης.

Τα βήματα για τη χρήση του Numeric Stepper είναι τα εξής:

1. Επιλέγουμε Window -> Components (αν δεν είναι ήδη ανοικτό) (Σχήμα 2.1.4)
2. Στο παράθυρο Components κάνουμε drag-n-drop το NumericStepper στο σημείο του animation που θέλουμε να το τοποθετήσουμε (Σχήμα. 2.1.5)



Σχήμα 2.1.4: Επιλογή του παράθυρου Components



Σχήμα 2.1.5: Τα περιεχόμενα του παράθυρου Components

2.1.5 Ανάθεση εντολών σε σύμβολα

Ένα χαρακτηριστικό της ActionScript 2 είναι η ανάθεση εντολών απ' ευθείας σε κάποιο σύμβολο (movie clip, button ή graphic) επιλέγοντάς το (με αριστερό κλικ) και γράφοντας απ' ευθείας στο παράθυρο Actions (το οποίο εμφανίζεται με το πλήκτρο F9 η επιλέγοντας Windows->Actions) το τι επιθυμούμε να γίνει όταν π.χ. κάνουμε αριστερό ή δεξί κλικ σε ένα button.

Δημιουργείται ένα symbol τύπου Button και στο οποίο ανατίθεται κάποια ενέργεια "action" με τον εξής τρόπο: Κάνοντας αριστερό κλικ στο σύμβολο ενεργοποιείται το πάνελ Actions – Button το οποίο θα δεχθεί την παρακάτω εντολή που το αφορά (Σχήμα 2.1.6).

```
on(release){
    gotoAndPlay(2);
    i=1;
}
```

Σχήμα 2.1.6: Η συνάρτηση on()

2.1.6 Η συνάρτηση on()

Η συνάρτηση on() ανήκει στην κατηγορία των handler functions και το όρισμα release δηλώνει πως μόλις γίνει κλικ στο συκεκριμένο button και αφεθεί το αριστερό πλήκτρο τότε θα εκτελεστούν οι εντολές που βρίσκονται μέσα στο μπλοκ.

*Υποσημείωση: Μια ενναλακτική μορφή θα ήταν να τοποθετηθεί ο παρακάτω κώδικας στο timeline του Stage:

```
myBtn.onRelease = function() {
  // Do something.
};
```

2.1.6 Η συνάρτηση gotoAndPlay()

Η συνάρτηση gotoAndPlay (2) μεταθέτει το playhead στο frame 2 και κάνει play. Επομένως, ο σκοπός των παραπάνω εντολών είναι να αρχίσει να «παίζει» το animation μόλις ο χρήστης κάνει release στο πλήκτρο start ή το πλήκτρο play (Σχήμα 2.1.7).

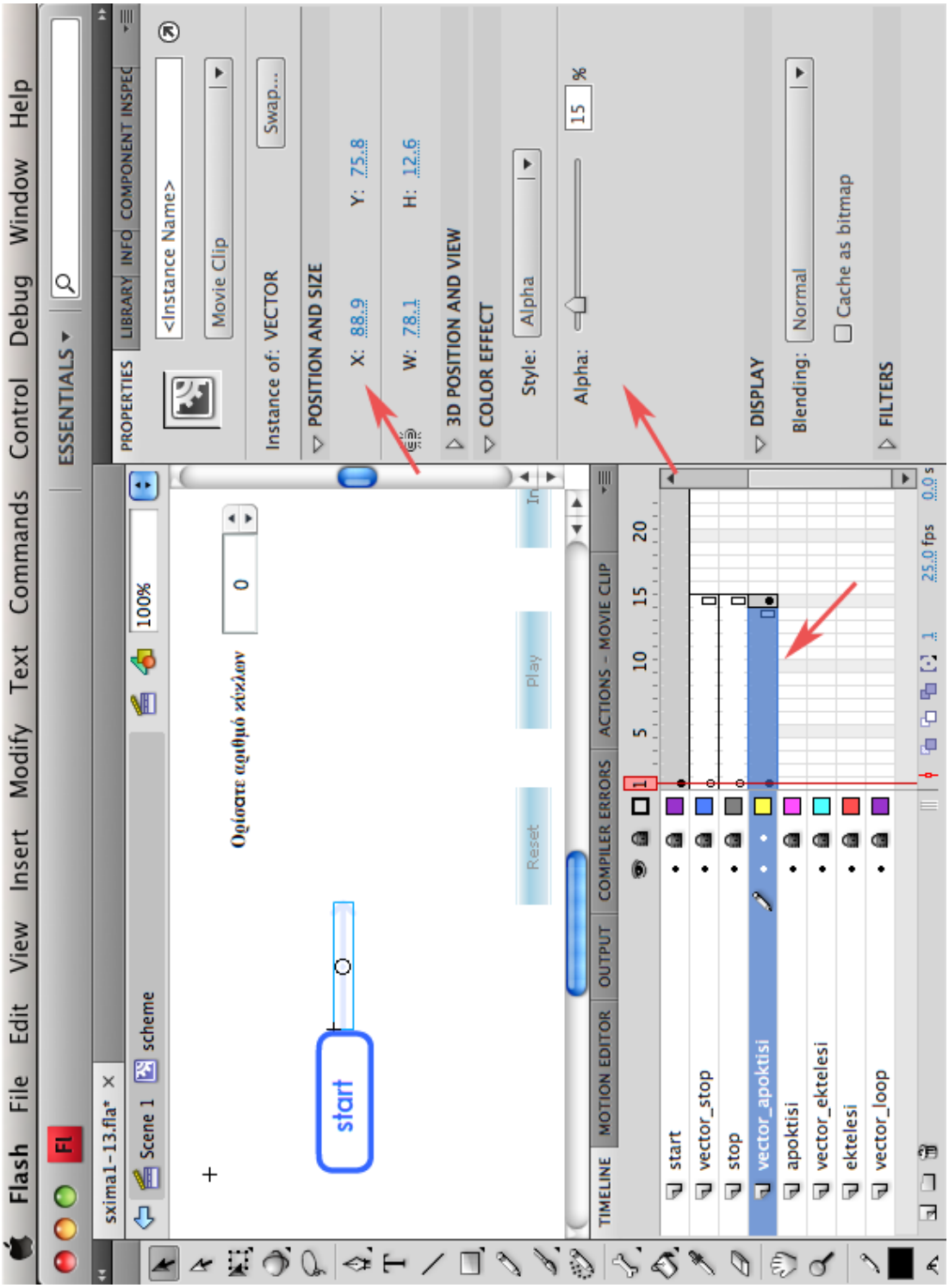
Σχήμα 2.1.7: Τα πλήκτρα start και play

Η μεταβλητή i υλοποιεί έναν counter όπου αυξάνει κατά 1 κάθε φορά για κάθε επανάληψη του animation μετρώντας έτσι το πόσες φορές έχει υλοποιηθεί ένας κύκλος. Στο συγκεκριμένο μπλοκ εντολών η μεταβλητή i αρχικοποιείται με την τιμή 1.

2.1.7 Δημιουργία Motion Tween

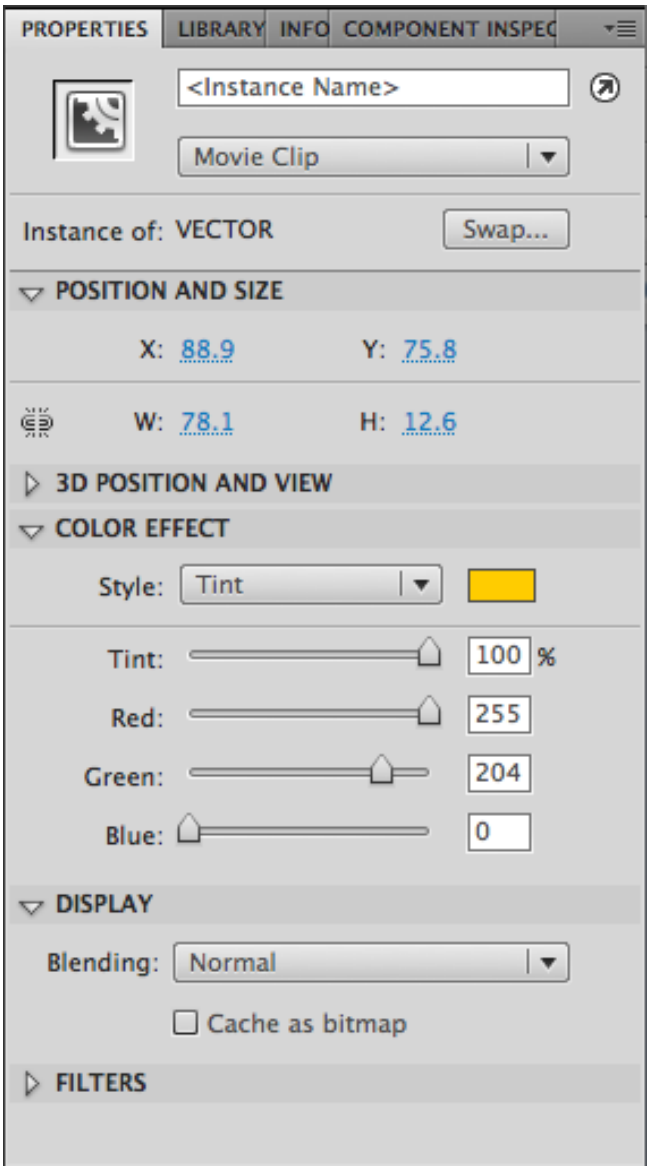
Ένας από τους τρόπους για την δημιουργία animation είναι η δημιουργία του Motion Tween. Αφού έχουν δημιουργηθεί τα σύμβολα που απεικονίζουν τα διανύσματα και τα υπόλοιπα βήματα του κύκλου απόκτησης – εκτέλεσης, γίνονται οι απαραίτητες ενέργειες για την αλλαγή του χρώματος από ανοιχτό μπλε σε κίτρινο με τον εξής τρόπο:

Επιλέγουμε το εκάστοτε σύμβολο (για παράδειγμα το διάνυσμα του layer vector_apokitisi) και δημιουργούμε από ένα keyframe στην αρχική (frame 1) και την τελική κατάσταση (frame 15). Αναλυτικότερα, στο frame 1, διαμορφώνουμε τις ιδιότητες όπως αυτές φαίνονται στην επόμενη εικόναν (Σχήμα 2.1.8).



Σχήμα 2.1.8: Ιδιότητες για την δημιουργία του Motion Tween

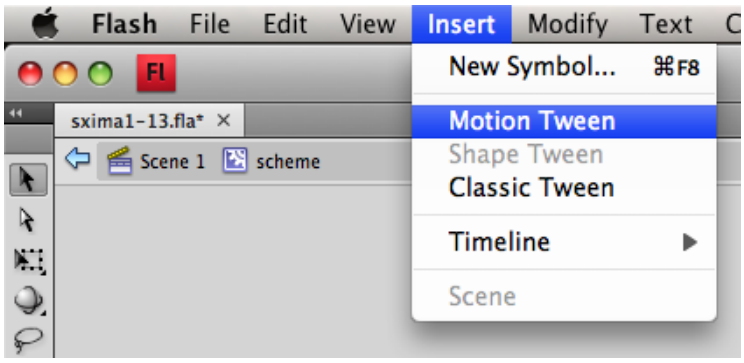
Στο frame 15 διαμορφώνουμε τις παρακάτω ιδιότητες (Σχήμα 2.1.9):



Σχήμα 2.1.9: Ρυθμίσεις της ιδιότητας Tint

Στη συνέχεια επιλέγουμε τα frames 1-15 και ενεργοποιούμε το Motion Tween. Η ενεργοποίηση του Motion Tween γίνεται με δύο τρόπους:

1. Επιλέγουμε Insert -> Motion Tween (Σχήμα 2.1.10) ή
2. Κάνουμε δεξί κλικ πάνω στα frames ή στο symbol επλέγουμε Motion Tween



Σχήμα 2.1.10: Ενεργοποίηση του Motion Tween

Με αυτή την επιλογή, το Flash δημιουργεί στο runtime και τα ενδιάμεσα frames (frames 2-14) προκειμένου να επιτευχθεί μία ομαλή μεταβολή των χρωμάτων. Όμοια διαμορφώνονται τα animation και για τα υπόλοιπα σχήματα.

*Υποσημείωση: Για την αποφυγή του φαινομένου να «σπάει» η κίνηση, το frame rate πρέπει να είναι μεγαλύτερο ή ίσο με 25 frames ανά δευτερόλεπτο.

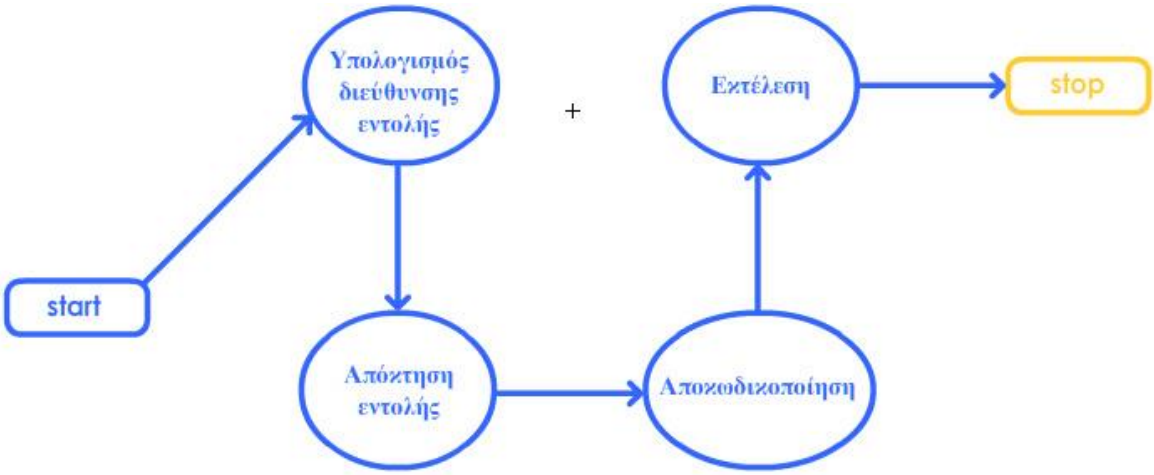
```
if(i!=k){
    gotoAndPlay(90);
}
else{
    gotoAndPlay(155);
}
```

Στο frame 89 έχει δημιουργηθεί μία συνθήκη if….else η οποία ελέγχει εάν η μεταβλητή i (η οποία περιέχει τον αριθμό του τρέχοντα κύκλου) είναι διαφορετική από την μεταβλητή k (περιέχει τον αριθμό των κύκλων που έχουμε ορίσει με το NumericStepper). Αν οι τιμές των μεταβλητών είναι διαφορετικές τότε, με την εντολή gotoAndPlay(90), το playback συνεχίζει κανονικά στο frame 90 όπου εκτελείται μέχρι και το frame 153 όπου σε εκείνο το frame μια άλλη εντολή η gotoAndPlay επαναλαμβάνει τον κύκλο από το frame 15 (και η μεταβλητή i αυξάνει κατά 1). Αν στο frame 89 οι τιμές των μεταβλητών i και k είναι ίσες, τότε το playback συνεχίζει από το frame 155 εφόσον εκτελείται η εντολή gotoAndPlay(155) και τότε το animation σταματάει στο frame 195 με την εντολή stop().

Με την χρήση των ανωτέρω στοιχείων του Flash και της ActionScript πραγματοιποείται ο βασικός κύκλος απόκτησης – εκτέλεσης. Στην επόμενη ενότητα θα αναλυθεί μια παραλλαγή του παραπάνω σχήματος.

2.2 Κύκλος απόκτησης – εκτέλεσης εντολής (β φάση)

Μια παραλλαγή του προηγούμενου παραδείγματος περιγράφεται στο παρακάτω σχήμα (Σχήμα 2.2.1).



Σχήμα 2.2.1: Κύκλος απόκτησης – εκτέλεσης εντολής (β φάση)

Έχουν προστεθεί δύο επιπλέον βήματα στον κύκλο απόκτησης – εκτέλεσης:

- Υπολογισμός διεύθυνσης εντολής
- Αποκωδικοποίηση.

Η διαδικασία υλοποίησης είνα παρόμοια με την υλοποίηση του προηγούμενου παραδείγματος, μερικές διαφορές όμως είναι οι παρακάτω:

Στο frame 179 γίνεται έλεγχος εάν η μεταβλητή i (η οποία περιέχει τον αριθμό του τρέχοντα κύκλου) είναι διαφορετική από την μεταβλητή k (περιέχει τον αριθμό των κύκλων που έχουμε ορίσει με το NumericStepper). Σε αυτή την περίπτωση το playhead συνεχίζει στο επόμενο frame με μία εναλλακτική συνάρτηση πλοήγησης, την nextFrame().

2.2.1 Οι συναρτήσεις nextFrame() και play()

Η συνάρτηση **nextFrame()** είναι διαθέσιμη από την έκδοση Flash 2 και μετά. Σκοπός είναι να μεταθέτει το playhead στο αμέσως επόμενο frame και να

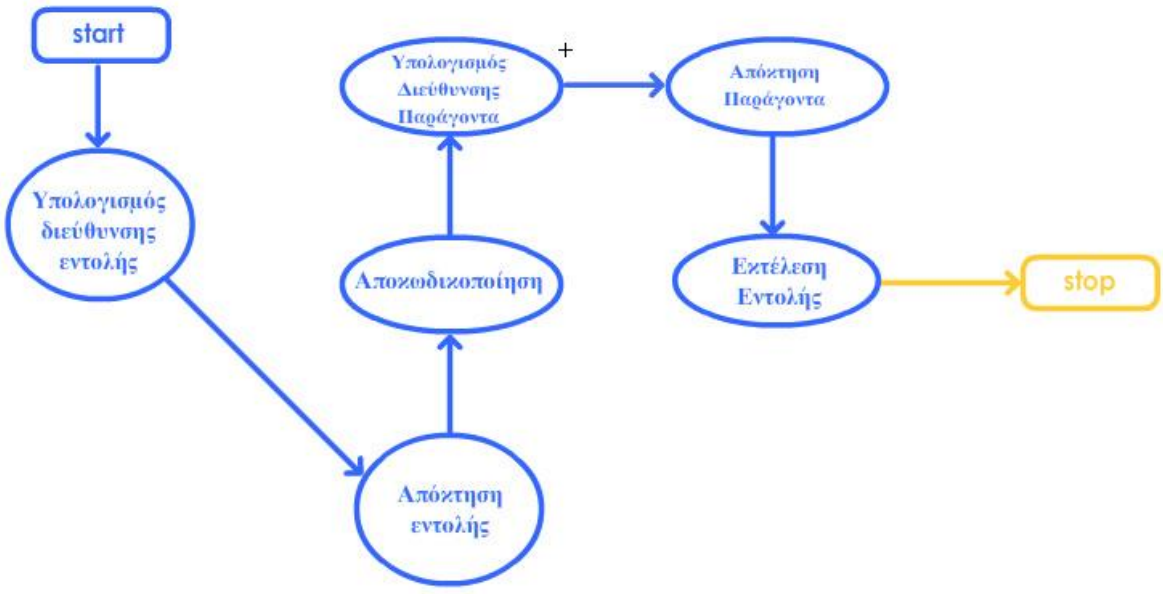
σταματάει. Για αυτό στην επόμενη σειρά προσθέτουμε την εντολή **play()** η οποία ξεκινάει το playhead (από το frame 180).

2.3. Κύκλος απόκτησης – εκτέλεσης εντολής (γ φάση)

Στο επόμενο παράδειγμα (Σχήμα 2.3.1) προστίθενται επιπλέον βήματα που σχετίζονται με τους παράγοντες εισόδου μιας εντολής, οι οποίοι σε κάποιες αρχιτεκτονικές μπορεί να είναι περισσότεροι από ένας. Τα νέα βήματα που προστίθενται είναι:

- ο υπολογισμός της διεύθυνσης των παραγόντων εισόδου
- η απόκτηση των παραγόντων.

Τα βήματα αυτά έπονται της αποκωδικοποίησης και προηγούνται της εκτέλεσης.



Σχήμα 2.3.1: Κύκλος απόκτησης – εκτέλεσης εντολής (γ φάση)

Για τον ορισμό του αριθμού των παραγόντων, γίνεται εισαγωγή ενός δεύτερου NumericStepper με όνομα paragontes. Στο frame 2 αποθηκεύονται οι τιμές των NumericStepper stepper (περιέχει τον αριθμό των κύκλων) και

paragontes (περιέχει τον αριθμό των παραγόντων) στις μεταβλητές k και p αντίστοιχα:

```
k=this.stepper.value;
p=this.paragontes.value;
```

2.3.1 Οι ιδιότητες this και value

Η ιδιότητα **this** ανήκει στις Global Properties, οι οποίες είναι διαθέσιμες και φαίνονται σε οποιοδήποτε σημείο του κώδικα, του timeline και γενικότερα σε οποιαδήποτε θέση του αρχείου flash. Σκοπός είναι να αναφέρει ένα αντικείμενο ή ένα στιγμιότυπο. Στις εντολές του frame 2 αναφέρεται στα NumericSteppers stepper και paragontes. Η ιδιότητα **value** αναφέρεται στην τρέχουσα τιμή που φαίνεται στην περιοχή κειμένου (text area) του component NumericStepper.

2.3.2 Υλοποίηση νέων βημάτων του κύκλου απόκτησης – εκτέλεσης εντολής

Τα στάδια υλοποίησης που αφορούν τα γραφικά και το Motion Tween είναι παρόμοια με τα προηγούμενα παραδείγματα. Το playhead του παραδείγματος αρχίζει με το release του πλήκτρου start ή του play. Στο frame 1 δηλώνεται μια νέα μεταβλητή, η j. Σκοπός της μεταβλητής j είναι να λειτουργεί ως ένας counter σχετικά με τα βήματα υπολογισμός διεύθυνσης παράγοντα – απόκτηση παράγοντα, τα οποία επαναλαμβάνονται για κάθε παράγοντα.

Στο frame 224 ελέγχεται αν η μεταβλητή j είναι διαφορετική από την μεταβλητή p. Ουσιαστικά η συνθήκη if…else ελέγχει αν ο τρέχων κύκλος που σχετίζεται με τον παράγοντα είναι ο τελευταίος ή όχι. Αν δεν είναι ο τελευταίος τότε το playhead πηγαίνει στο frame 330 με την εντολή gotoAndPlay(330) ενώ η μεταβλητή j αυξάνει κατά 1.

```
if(j!=p) {
  gotoAndPlay(330);
  j++;
}
```

Σε αυτή την περίπτωση το playhead παίζει μέχρι το frame 415 όπου εκεί η εντολή gotoAndPlay(224) επαναφέρει το playback στο frame 224. Στην περίπτωση που η τιμή της μεταβλητής j είναι ίση με την τιμή της μεταβλητής p – δηλαδή ο τρέχων κύκλος που αφορά τον παράγοντα είναι και ο τελευταίος- το playback συνεχίζει να παίζει μέχρι και το frame 271 όπου εκεί γίνεται έλεγχος για τον αν ο τρέχων κύκλος απόκτησης – εκτέλεσης είναι και ο τελευταίος. Αν δεν είναι ο τελευταίος κύκλος τότε με την εντολή gotoAndPlay(3) το playback μεταφέρεται στο frame 3 για την αναπαράσταση του επόμενου κύκλου.

```
if(i!=k){
  gotoAndPlay(3);
  i++;
}
```

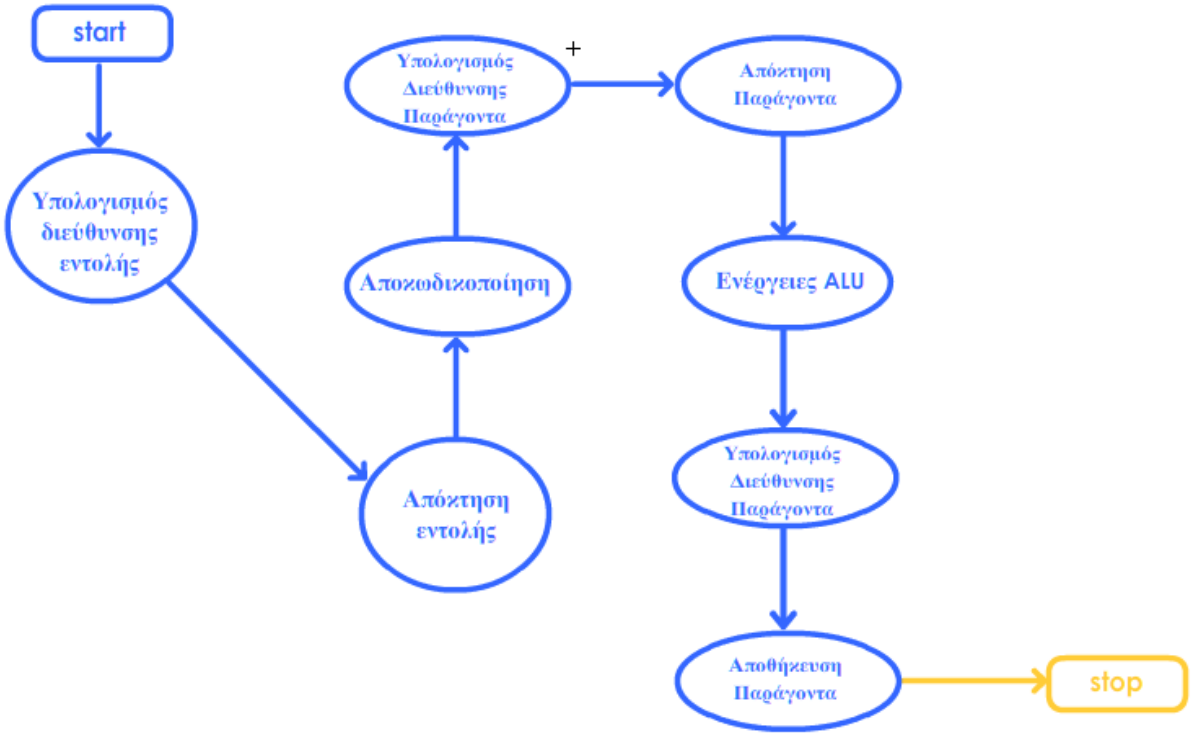
Διαφορετικά, αν ο κύκλος είναι ο τελευταίος, τότε το playback συνεχίζει μέχρι και το frame 312 όπου και τερματίζει με την εντολή stop().

2.4 Τελικός κύκλος απόκτησης – εκτέλεσης εντολής

Στην επόμενη παραλλαγή του κύκλου Απόκτησης - Εκτέλεσης Εντολής (Σχήμα 2.4.1) προστίθεται η επιλογή του παράγοντα εξόδου και επομένως τα βήματα:

- Υπολογισμός Διεύθυνσης Παράγοντα
- Αποθήκευση Παράγοντα

τα οποία σχετίζονται με τον παράγοντα εξόδου.



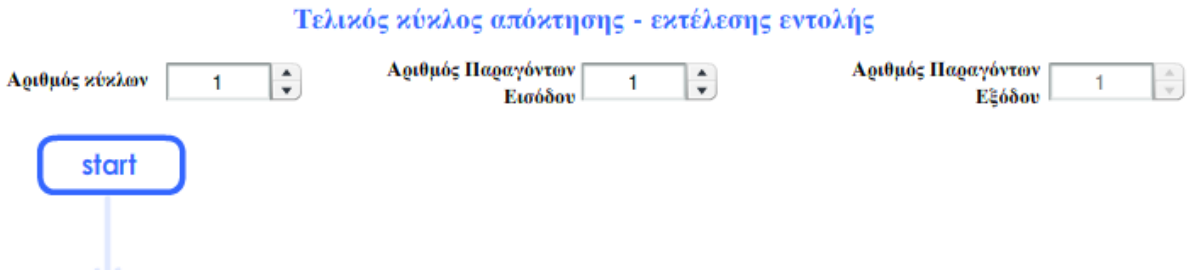
Σχήμα 2.4.1: Τελικός κύκλος απόκτησης – εκτέλεσης εντολής

Για τον ορισμό του αριθμού των παραγόντων εξόδου, γίνεται εισαγωγή ενός τρίτου NumericStepper με όνομα paragontes_eks. Στο frame 2 αποθηκεύεται και η τιμή του NumericStepper paragontes_eks στη μεταβλητή s (Σχήμα 2.4.2)

```
1  k=this.stepper.value;
2  p=this.paragontes.value;
3  s=this.paragontes_eks.value;
4
```

Σχήμα 2.4.2: Ανάθεση τιμής ενός NumericStepper σε μεταβλητή

Επομένως η αρχική κατάσταση του σχήματος είναι η ακόλουθη (Σχήμα 2.4.3):



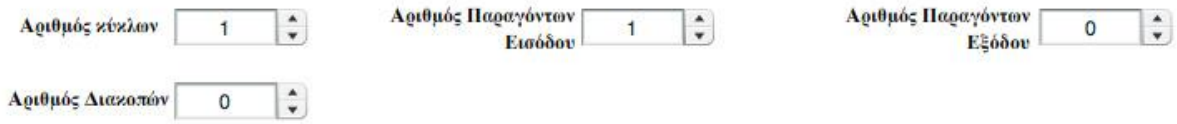
Σχήμα 2.4.3: Αρχική οθόνη του τελικού κύκλου

Αντίστοιχα, ο έλεγχος του αριθμού των κύκλων γίνεται στο frame 523 όπου ελέγχεται αν ο τρέχων κύκλος είναι ο τελευταίος προκειμένου το playback να συνεχίσει μέχρι το τέλος του animation, διαφορετικά να μεταφερθεί στο frame 565 και στο frame 595 με την εντολή gotoAndPlay(40) μεταφερθεί στο frame 40 για να εκτελεστεί ο επόμενος κύκλος.

2.5 Κύκλος απόκτησης – εκτέλεσης εντολής με διακοπή

Στην τελευταία παραλλαγή του Κύκλου Απόκτησης – Εκτέλεσης εντολής προστίθεται το βήμα ελέγχου για την ύπαρξη ή όχι διακοπής.

Άλλη μια επιλογή που προστίθεται στην αρχική οθόνη είναι ο προσδιορισμός του αριθμού των διακοπών (Σχήμα 2.5.1)



Σχήμα 2.5.1: Προσθήκη Combo Box για αριθμό διακοπών

Με αυτή την επιλογεί καθορίζουμε τον αριθμό των διακοπών που θα συμβούν στο σύνολο των Κύκλων – Απόκτησης Εκτέλεσης εντολής.

Επομένως, ο αριθμός των διακοπών θα πρέπει να είναι μικρότερος από το πλήθος των κύκλων. Γι' αυτό στο frame 2 γίνεται έλεγχος αν η τιμή του αριθμού των διακοπών είναι μικρότερος προκειμένου να γίνει εκκίνηση του animation, διαφορετικά εμφανίζεται το μήνυμα:

«Το πλήθος των διακοπών πρέπει να είναι μικρότερο από το πλήθος των κύκλων.»

προκειμένου ο χρήστης να επιλέξει μια έγκυρη τιμή.

Το combo box που σχετίζεται με την επιλογή των διακοπών έχει χαρακτηριστικό όνομα break_stepper και η μεταβλητή breaks φέρει την τιμή του.

Στην συνέχεια γίνεται επεξήγηση του τρόπου με τον οποίο οι διακοπές ανατίθενται στους κύκλους. Αυτό επιτυγχάνεται με τον παρακάτω κώδικα (Σχήμα 2.5.2)

```
var arrayRandom:Array = new Array();
var low = 1;
var high = k;
var random_temp;

Array.prototype.indexOf = function(obj):Number  {
    for (var p:Number = 0; p < this.length; p++)
    {
        if (this[p] == obj) return p;
    }
        return -1;
    };

if(k>1 && breaks >0){
    for(var p=0; p<breaks; p++){
        do
        {
            random_temp = Math.round(Math.random() * (high - low)) + low;
        }
        while((arrayRandom.indexOf(random_temp) > -1) || random_temp == high);

            arrayRandom[p] = random_temp;

    }
    arrayRandom.sort();
```

Σχήμα 2.5.2: Κώδικας ανάθεσης διακοπών

2.5.1 Συναρτήσεις Math.round() και Math.random()

Αρχικά δημιουργείται ένας πίνακας με όνομα arrayRandom και οι μεταβλητές low (με τιμή 1), high (με τιμή των αριθμό των κύκλων) και random_temp. Το υπόλοιπο τμήμα κώδικα έχει σκοπό να παράγει με τη χρήση της συνάρτησης **Math.random()** τυχαίες τιμές μεταξύ των τιμών των μεταβλητών low και high που θα αντιπροσωπεύουν τους κύκλους στους οποίους θα συμβούν διακοπές. Η συνάρτηση **Math.round()** στρογγυλοποιεί τις τυχαίες τιμές προς τον κοντινότερο ακέραιο αριθμό.

Στη συνέχεια αυτές οι τιμές αποθηκέυονται μία προς μία στον πίνακα arrayRandom και τέλος ο πίνακας arrayRandom ταξινομείται κατά αύξουσα σειρά με τη χρήση της συνάρτησης sort(). Έτσι, αν για παράδειγμα έχουμε επιλέξει αριθμό κύκλων 5 και αριθμό διακοπών 2, τότε η συνάρτηση random θα παράγει δύο ακέραιους αριθμούς π.χ. τον 3 και τον 1, θα τους αποθηκέυσει στον πίνακα, στην συνέχεια θα ταξινομήσει τον πίνακα προκειμένου να είναι με την σειρά 1, 3 και στο runtime θα συμβούν διακοπές στον κύκλο 1 και στον κύκλο 3 αντίστοιχα.

2.5.1 Συνάρτηση index.Of()

Η τρόπος υλοποίησης είναι ο ίδιος μέχρι και το βήμα Αποθήκευση παράγοντα. Από εκεί και μετά το playback μεταφέρεται στο frame 565, όπου εμφανίζεται διαδοχικά το βήμα «Έλεγχος διακοπής». Στο frame 609 έχει συνταχθεί η εξής εντολή:

```
if(arrayRandom.indexOf(i) != -1){ gotoAndPlay(670); }
```

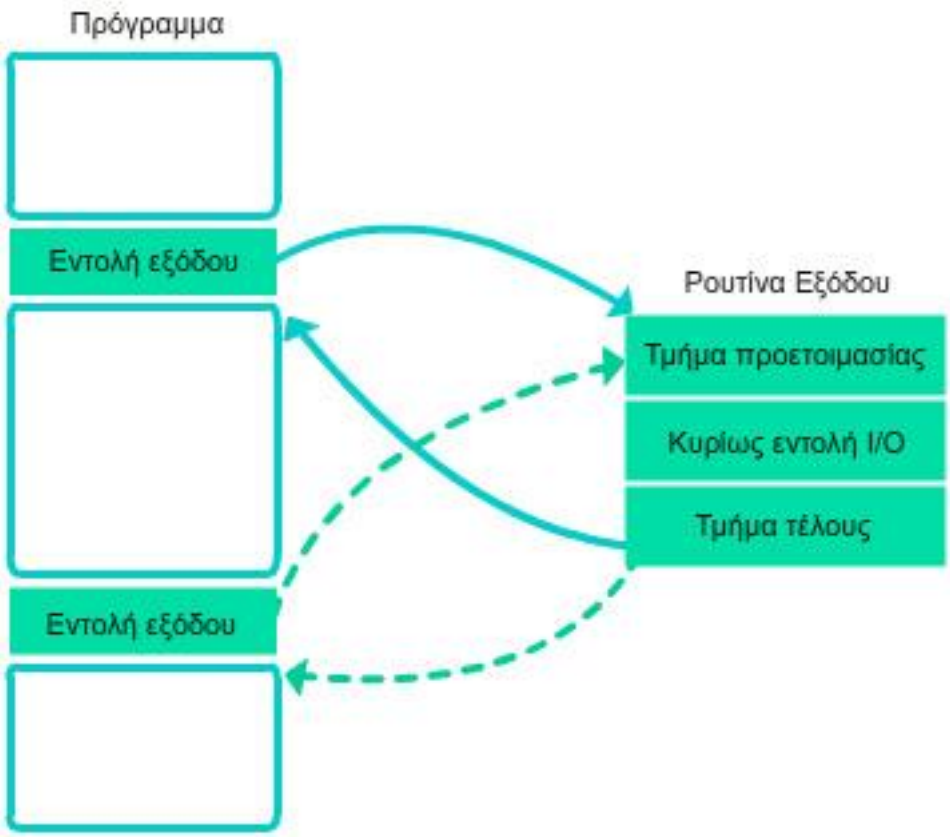
Η συνάρτηση indexOf(i) κάνει αναζήτηση στον πίνακα για την τιμή που δέχεται ως όρισμα και επιστρέφει τον αριθμό της θέσης του πίνακα. Αν δεν βρει την τιμή επιστρέφει τον αριθμό -1. Επομένως, στην παραπάνω εντολή (η μεταβλητή i περιέχει την τιμή του τρέχοντα κύκλου) γίνεται έλεγχος αν στον τρέχοντα κύκλο θα συμβεί διακοπή. Αν ναι, τότε το playback μετατίθεται στο frame 670 όπου εκτελείται το τμήμα διακοπής, διαφορετικά συνεχίζει στο επόμενο frame όπου εκτελείται ο επόμενος κύκλος.

ΚΕΦΑΛΑΙΟ 3 – Παραδείγματα σε ActionScript 3.0

3.1 Εκτέλεση εντολής εξόδου χωρίς διακοπή

Στο επόμενο παράδειγμα που αφορά την εκτέλεση εντολής εξόδου χωρίς διακοπή (Σχήμα 3.1.1), θα επισημανθούν και θα αναπτυχθούν τα παρακάτω χαρακτηριστικά:

- addEventListener
- Timer
- Import
- Tween
- Color
- isPlaying
- try…catch
- events
- Functions
- event.target.name
- switch…case
- Arrays



Σχήμα 3.1.1: Εκτέλεση εντολής εξόδου χωρίς διακοπή

Ο τρόπος υλοποίησης είναι διαφορετικός σε σχέση με τα προηγούμενα παραδείγματα καθώς γίνεται εκτενής χρήση της ActionScript 3.0.

3.1.1 Η μέθοδος addEventListener()

Ο χειρισμός των Buttons δεν γίνεται με την απευθείας ανάθεση κάποιων actions πάνω στο Button (όπως γινόταν με την ActionScript 2.0), αλλά γίνεται με τη χρήση της μεθόδου addEventListener. Τα Event Listeners, τα οποία λέγονται και event handlers, είναι συναρτήσεις που εκτελούνται ως αποτέλεσμα κάποιου event. Η προσθήκη ενός event listener είναι μια διαδικασία δύο βημάυων:

1. Δημιουργία της συνάρτησης που θα εκτελεστεί ως ανταπόκριση κάποιου event. Η συνάρτηση αποκαλείται listener function ή event handler function.

2. Χρήση της μεθόδου addEventListener() προκειμένου να γίνει η αντιστοίχιση του event με την συνάρτηση που δημιουργήθηκε στο πρώτο βήμα.

Η μέθοδος addEventListener ανήκει στην κλάση IEventDispatcher που κληρονομεί την Object. Χρησιμοποιείται για την ανάθεση των listener functions. Οι δύο απαιτούμενες παράμετροι ή ορίσματα είναι 1) type και 2) listener. Η παράμετρος type χρησιμοποιείται για τη διευκρίνιση του είδους του event. Π.χ. Η παράμετρος MouseEvent.MOUSE_OVER δηλώνει ότι η συνάρτηση listener θα ενεργοποιηθεί όταν ο κέρσορας του Mouse «περάσει» πάνω από κάποιο αντικείμενο. Η παράμετρος listener καλεί την listener function.

*Υποσημείωση: Στην παράμετρο listener ΔΕΝ χρησιμοποιούνται παρενθέσεις για τη δήλωση τις listener function. Π.χ. η συνάρτηση clickHandler() δηλώνεται χωρίς παρενθέσεις στην μέθοδο addEventListener():
addEventListener(MouseEvent.CLICK, **clickHandler**)

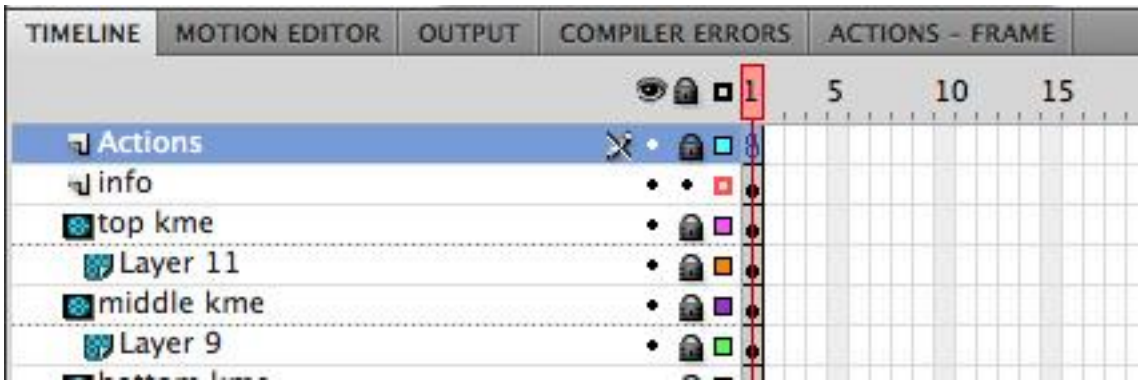
Στην υλοποίηση του παραδείγματος αρχικά έχει γίνει χρήση της μεθόδου addEventListener() στους controllers που σχετίζονται με την εκκίνηση και την παύση του animation.

Πιο συγκεκριμένα, για το πλήκτρο play 

αναθέτουμε την παρακάτω εντολή:

```
this.controllers.znext.addEventListener(MouseEvent.CLICK,eventHandler);
```

Αξίζει να επισημανθεί πως η εντολή δεν έχει γραφτεί πάνω στο symbol (γιατί έχει συνταχθεί σε AS3), αλλά έχει δημιουργηθεί ένα ξεχωριστό layer με διακριτό όνομα Actions όπου στο πρώτο keyframe έχει συνταχθεί ο πλήρης κώδικας του παραδείγματος (Σχήμα 3.2.2).



Σχήμα 3.1.2: Δημιουργία Layer για την σύνταξη των Actions

Με την παραπάνω εντολή καλούμε την συνάρτηση evenHandler όταν το Mouse κάνει Click πάνω στο πλήκτρο Play (είναι το instance znext).

Με τον ίδιο τρόπο συντάσεται και η εντολή που αφορά το πλήκτρο reset (είναι το instance zstop):

this.controllers.zstop.addEventListener(MouseEvent.CLICK,eventHandler);

3.1.2 Υλοποίηση συνάρτησης eventHandler()

Η συνάρτηση eventHandler() (Σχήμα 3.1.3) εντοπίζει το όνομα του instance που προκάλεσε το event και κατόπιν κατευθύνει τη ροή του animation προς τη σχετική ενέργεια. Το όρισμα που δέχεται η συνάρτηση είναι τύπου MouseEvent και έχει όνομα event. Όταν προκύπτει κάποιο event, η μέθοδος addEventListener() επιστρέγει το είδος του σχετικού event που προκλήθηκε.

Για παράδειγμα,= πατώντας το πλήκτρο Play, επιστρέφεται από τη μέθοδο addEventListener() το παρακάτω μήνυμα:

[MouseEvent type="click" bubbles=true cancelable=false eventPhase=2 localX=23 localY=7 stageX=303 stageY=462 relatedObject=null ctrlKey=false altKey=false shiftKey=false buttonDown=false delta=0]

```
function eventHandler(event:MouseEvent):void {

    switch (event.target.name){
        case "znext":
            controller("step01");
            this.controllers.znext.mouseEnabled=false;
        break;
        case "zstop":
            freeze(event.target.name);
        break;
    }
}
```

Σχήμα 3.1.3: Συνάρτηση eventHandler()

Με μια συνθήκη τύπου switch… case εντοπίζεται το instance που προκάλεσε το event και κατόπιν εκτελείται το τμήμα κώδικα του σχετικού case.

Η ιδιότητα event.target.name επιστρέφει το όνομα (τύπου String) του στιγμιότυπου (instance) που προκάλεσε το event.

Αν το instance είναι το **znext** τότε καλείται η συνάρτηση controller() η οποία θα αναλυθεί παρακάτω. Παράλληλα, ορίζοντας ως false την ιδιότητα (property) mouseEnabled απενεργοποιείται το πλήκτρο **znext** για την αποφυγή ανεπιθύμητης διακοπής του animation και εκκίνησής του από την αρχή.

Αν το instance είναι το **zstop** τότε καλείται η συνάρτηση freeze() με την οποία επιτυγχάνεται η διακοπή του animation και η αρχικοποίησή του reset. Η συνάρτηση freeze() θα αναλυθεί παρακάτω.

Στη συνέχεια θα γίνει ανάλυση της συνάρτησης controller().

3.1.3 Υλοποίηση συνάρτησης controller()

```
function controller(steps:String):void {

    switch (steps){
        case "step01":
            top_panel_tween  = new  Tween(this.top_panel, "y",None.easeNone,4,67,3,true);
            top_panel_tween.addEventListener("motionFinish", entoli_eksodou_anim);
            break;
        case "step02":
            this.entoli_eksodou_01.transform.colorTransform  = cYellow;
            myTimerArray[1].addEventListener(TimerEvent.TIMER, pros_tmima_proet);
            myTimerArray[1].start();
            break;
              .
              .
              .
              .
    }
}
```

Σχήμα 3.1.4: Συνάρτηση controller()

Στο Σχήμα 3.1.4 υπάρχουν ενδεικτικά τα δύο πρώτα βήματα. Αρχικά, η συνάρτηση controller δέχεται ένα όρισμα (steps) τύπου String το οποίο είναι το όνομα του βήματος που θέλουμε να εκτελεστεί. Επισημαίνεται πως η υλοποίηση του παραδείγματος αποτελείται από 11 επιμέρους βήματα.

Με το πάτημα του πλήκτρου 

η συνάρτηση eventHandler καλεί την συνάρτηση controller και της μεταδίδει το όρισμα “Step01” προκειμένου να γίνει εκκίνηση του animation από την αρχή. Με μια συνθήκη τύπου Switch…case έχουν τμηματοποιηθεί οι σχετικές ενέργειες των βημάτων 1-11.

Πριν προχωρήσουμε σε ποιο διεξοδική ανάλυση χρειάζεται να επεξηγηθούν κάποια προηγούμενα βήματα. Το πρώτο βήμα, αφορά τη χρήση των κλάσεων Timer, TimerEvent, Sprite, Color και Tween καθώς και των κλάσεων των πακέτων flash.display και fl.transitions.easing. Προκειμένου να χρησιμοποιηθούν οι αντίστοιχες μέθοδοι των παραπάνω κλάσεων έχει γίνει εισαγωγή αυτών των κλάσεων με τη χρήση της εντολής import (Σχήμα 3.1.5)

```
import flash.utils.Timer;
import flash.events.TimerEvent;
import flash.display.Sprite;
import fl.motion.Color;
import flash.display.*;
import fl.transitions.Tween;
import fl.transitions.easing.*;
```

Σχήμα 3.1.5: Χρήση εντολής import

3.1.4 Εισαγωγή πακέτων και κλάσεων

Στην ActionScript 3, η εντολή import είναι απαραίτητη για την πρόσβαση των κλάσεων. Ο wildcard character (*) χρησιμοποιείται για την εισαγωγή όλων των κλάσεων ενός πακέτου.

3.1.5 Η κλάση Timer

Η κλάση **Timer** ανήκει στο πακέτο flash.utils και περιέχει όλες τις απαραίτητες μεθόδους για το χρονισμό τμημάτων κώδικα. Δημιουργούνται αντικείμενα τύπου Timer τα οποία μπορούν να εκτελεστούν μια φορά ή κατ' επανάληψη προκειμένου να εκτελεστεί το σχετικό τμήμα κώδικα μετά από συγκεκριμένο μήκος χρόνου. Χρειάζεται η προσθήκη ενός event listener ο οποίος θα ενεργοποιήσει το τμήμα κώδικα μετά το πέρας του χρόνου. Η υλοποίηση του τελευταίου γίνεται με τη χρήση ενός αντικειμένου TImerEvent της κλάσης **TimerEvent** του πακέτου flash.events.

3.1.6 Η κλάση Sprite

Η κλάση **Sprite** χρησιμοποιείται για την υλοποίηση γραφικών τα οποία δεν έχουν δικό τους timeline. Ανήκει στο πακέτο flash.display.

3.1.7 Η κλάση Color

Η κλάση **Color** ανήκει στο πακέτο fl.motion και επεκτείνει την κλάση ColorTransform. Προσθέτει τη δυνατότητα ελέγχου του brightness και tint.

3.1.8 Η κλάση Tween

Η κλάση **Tween** ανήκει στο πακέτο fl.transitions και δίνει τη δυνατότητα μετακίνησης, αλλαγής μεγέθους ή βαθμιαίου σβησίματος (fade) με τη χρήση της Actionscript. Αυτό επιτυγχάνεται με την ανάθεση συγκεκριμένων ιδιοτήτων στα movie clips οι οποίες ιδιότητες αλλάζουν τιμή μετά το πέρας κάποιων frames ή δευτερολέπτων. Για τη χρήση των μεθόδων και των ιδιοτήτων της κλάσης Tween, απαιτείται η δημιουργία ενός στιγμιότυπου τύπου Tween και στη συνέχεια επιλέγουμε την ανάλογη μέθοδο.

3.1.9 Το πακέτο fl.trasnition.easing

Το πακέτο fl.transition.easing περιλαμβάνει κλάσεις που μπορούν να χρησιμοποιηθούν για την επιτάχυνση ή την επιβράδυνση ενός effect, προκειμένου το animation να γίνει ποιο ρεαλιστικό στην κίνηση. Οι κλάσεις που περιλαμβάνει είναι:

- Back
- Bounce
- Elastic
- None
- Regular
- Strong

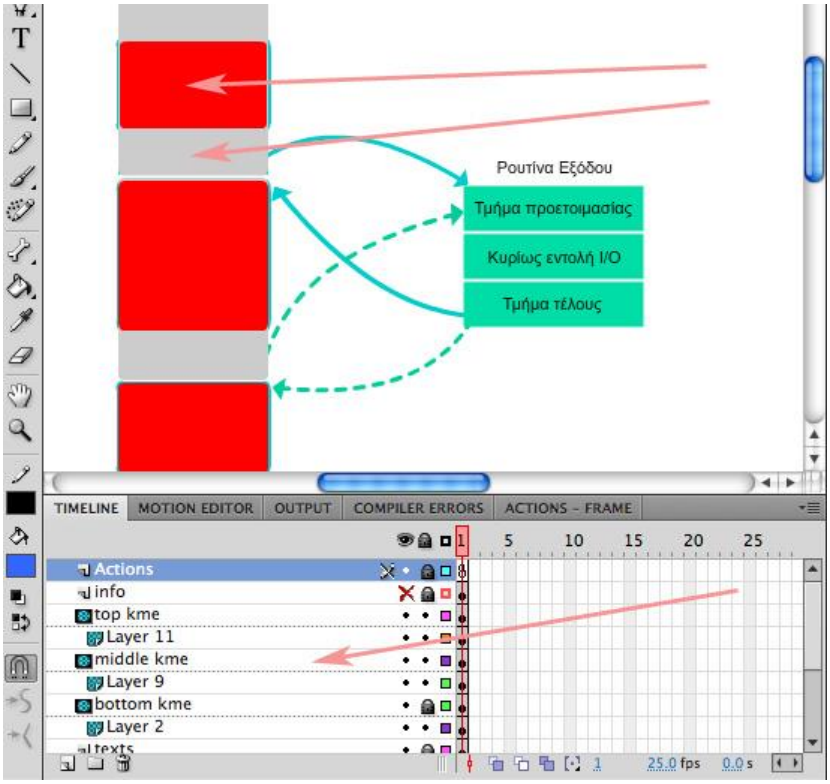
Στην υλοποίηση του παραδείγματος θα εξετάσουμε μερικές από αυτές.

3.1.10 Τεχνική μάσκας

Μια ακόμη επισήμανση αφορά τη χρήση μάσκας. Για τη δημιουργία δυναμικών εφέ συνιστάται η χρήση μάσκας. Η μάσκα πρακτικά λειτουργεί σαν ένα «παράθυρο» που αποκρύπτει τα layers που βρίσκονται εκτός της περιοχής της μάσκας και εμφανίζει μόνο τα layers που βρίσκονται εντός της περιοχής.

Ως μάσκα μπορεί να χρησιμοποιηθεί ένα γραφικό, ένα movie clip ή οποιοδήποτε άλλο στιγμιότυπου (instance). Μπορούν πολλά layers να γίνουν ομάδα μιας μάσκας.

Στο παράδειγμα έχει γίνει χρήση μάσκας προκειμένου να επιτευχθεί το animation που αφορά την «εκτέλεση» του κώδικα στην ΚΜΕ. Όπως φαίνεται στο Σχήμα 3.1.6 έχουν δημιουργηθεί τρία layer-μάσκες με αντίστοιχα ονόματα: top kme, middle kme και bottom kme, τα οποία στο stage φαίνονται με έντονο κόκκινο. Ενώ κάτω από αυτά βρίσκεται από ένα Layer τα οποία περιλαμβάνουν από ένα γκρι panel το οποίο έχει αντίστοιχα instance names: top_panel, middle_panel και bottom_panel.



Σχήμα 3.1.6: Απεικόνιση των mask layer

3.1.11 Δημιουργία αντικειμένου τύπου Tween

Πατώντας το πλήκτρο Play η συνάρτηση controller() έχει δεχθεί ως όρισμα το «step01». Η πρώτη εντολή στο case “step01” είναι η:

```
top_panel_tween = new Tween(this.top_panel,
“y",None.easeNone,4,67,3,true);
```

Στην αρχή του κώδικα έχει δημιουργηθεί μια αναφορά τύπου Tween με όνομα top_panel_tween. Με την παραπάνω εντολή δημιουργείται ένα αντικείμενο τύπου Tween το οποίο θα μετακινήσει το instance top_panel στον άξονα «y» με ιδιότητα κίνσης None (δηλ. Ομαλή κίνηση) από τη θέση 4 στη θέση 67 σε 3 δευτερόλεπτα. Το όρισμα true διευκρινίζει ότι η μέτρηση του χρόνου είναι σε δευτερόλεπτα. Αν ήταν false τότε ο αριθμός 3 θα σήμαινε 3 frames. Η αμέσως επόμενη εντολή είναι η:

```
top_panel_tween.addEventListener("motionFinish", entoli_eksodou_anim);
```

Χρησιμοποιείται η μέθοδος addEventListener προκειμένου μόλις τελειώσει η κίνηση (**motionFinish**) να μεταφερθεί η ροή του προγράμματος στη συνάρτηση entoli_eksodou_anim. Η συνάρτηση entoli_eskodou_anim περιέχει την παρακάτω εντολή:

```
function entoli_eksodou_anim(event:Event):void {
  controller("step02");
}
```

Καλεί εκ νέου τη συνάρτηση controller δίνοντας ως όρισμα την τιμή step02 προκειμένου το animation να συνεχίσει στο επόμενο βήμα.

3.1.12 Δημιουργία αντικειμένου τύπου Color και οι βασικές του ιδιότητες

Στη συνάρτηση controller στο τμήμα που αφορά το 2^{ο} βήμα η πρώτη εντολής είναι:

```
this.entoli_eksodou_01.transform.colorTransform = cYellow;
```

Το αντικείμενο cYellow είναι τύπου Color και έχει δημιουργηθεί στην αρχή του κώδικα με τις παρακάτω εντολές:

```
var cYellow:Color=new Color();
cYellow.setTint(0xffdc51, 1);
```

Η μέθοδος **setTint()** ρυθμίζει τις παραμέτρους που αφορούν την ιδιότητα Tint ενός συμβόλου. Τα δύο ορίσματα που δέχεται είναι το χρώμα (σε 16δική μορφή) και το alpha(το ποσοστό εμφάνισης). Η τιμή 1 είναι η μεγαλύτερη δυνατή λέγοντας πως το symbol θα γίνει κίτρινο 100%. Αν η τιμή ήταν 0.5 το ποσοστό κίτρινου χρώματος θα ήταν 50% κοκ.

Επομένως το στιγμιότυπο entoli_eksodou_01 θα γίνει κίτρινο μόλις τελειώσει η κίνηση του top_panel.

3.1.13 Δημιουργία αντικειμένου τύπου Timer

Οι δύο επόμενες εντολές είναι:

```
myTimerArray[1].addEventListener(TimerEvent.TIMER, pros_tmima_proet);
myTimerArray[1].start();
```

Ο πίνακας myTimerArray περιέχει αντικείμενα τύπου Timer τα οποία χρησιμεύουν για το χρονισμό του animation και έχει δημιουργηθεί με το παρακάτω τμήμα κώδικα:

```
var myTimerArray:Array = new Array();
 for (var i:int=1; i<10; i++) {
  var myTimer:Timer = new Timer(1000, 1);
  myTimerArray[i] = myTimer;
}
```

Επομένως μόλις το σύμβολο entoli_eksodou_01 γίνει κίτρινο το animation θα περιμένει 1 δευτερόλεπτο για την εκτέλεση της επόμενης εντολής και ποιο συγκεκριμένα της εκτέλεσης του κώδικα της συνάρτησης pros_tmima_proet.

Με τον ίδιο τρόπο εκτελούνται ένα προς ένα τα βήματα μέχρι και την ολοκλήρωση του animation.

3.1.14 Ανάλυση της συνάρτησης freeze()

Στη συνέχεια γίνεται ανάλυση της συνάρτησης **freeze().** Η συνάρτηση freeze() καλείται όταν επιλεγεί το πλήκτρο reset με σκοπό να σταματήσει το animation και να μεταβεί στην αρχική του κατάσταση.

Η πρώτη εντολή της συνάρτησης freeze() είναι η:

```
this.controllers.znext.mouseEnabled=true;
```

Η παραπάνω εντολή ενεργοποιεί και πάλι το πλήκτρο Play (το οποίο έχει το όνομα znext) προκειμένου να δοθεί η επιλογή εκκίνησης του animation.

Αμέσως μετά καλείται η συνάρτηση cBlankInit() όπου σκοπός της είναι να αποχρωματίζει όσα panel έχουν χρώμα κίτρινο. Αυτό επιτυγχάνεται με την εντολή

```
this.[OBJECT].transform.colorTransform = cBlank;
```

Όπως έχει επεξηγηθεί γίνεται αλλαγή της τιμής της ιδιότητας **colorTransform** σε τιμή cBlank όπου η τιμή αυτή είναι ένα αντικείμενο τύπου **Color** με τιμές που αφορούν το Tint (0xffffff, 0).

Στη συνέχεια γίνεται έλεγχος προκειμένου να προσδιοριστεί σε ποιο χρονικό σημείο του animation έγινε η διακοπή. Έτσι, με τον παρακάτω κώδικα γίνεται έλεγχος αν το πάνω panel (top_panel) είναι εν κινήσει. Η ιδιότητα **isPlaying** είναι τύπου Boolean και επιστρέφει true στην περίπτωση που κάποιο αντικείμενο τύπου Tween “παίζει” ή όχι.

```
if(this.top_panel_tween.isPlaying)
```

```
{
        this.top_panel_tween.stop();
        this.top_panel.y = -100;
}
```

Όταν λοιπόν το αντικείμενο top_panel_tween βρίσκεται εν κινήσει τότε με την εντολή stop() γίνεται παύση του Tween ενώ η επόμενη εντολή μεταθέτει το top_panel στην θέση -100 στον άξονα y προκειμένου να γίνει αρχικοποίηση της κατάστασής του.

Το επόμενο τμήμα κώδικα ελέγχει αν η κίνηση (tween) του πάνω panel (top_panel) έχει τερματιστεί:

```
else if(this.top_panel_tween.finish==67){
        this.top_panel.y      = -100;
        this.middle_panel.y  = -100;
        this.bottom_panel.y = -100;
        for (var i:int=1; i<10; i++) {
                myTimerArray[i].stop();
        }
}
```

Η ιδιότητα **finish** επιστρέφει έναν αριθμό, ο οποίος αντιπροσωπεύει την τελική τιμή της θέσης του σχετικού αντικειμένου που έχει κινηθεί. Πιο συγκεκριμένα, το top_panel σταματάει την κίνησή του στην θέση 67 του άξονα y. Στην περίπτωση που το top_panel έχει τελειώσει την κίνησή του, τότε αρχικοποιούνται και τα υπόλοιπα panel στην αρχική τους θέση. Με το loop γίνεται και παύση του χρονισμού που αφορά τα αντικείμενα τύπου Timer και βρίσκονται στον πίνακα myTimerArray. Σε προηγούμενη ενότητα έχει γίνει επεξήγηση της ανάπτυξής των.

Παρόμοιος έλεγχος γίνεται και για την κίνηση των επόμενων γκρι panels προκειμένου να αρχικοποιηθούν στην αρχική τους θέση.

3.1.15 Χειρισμός εξαιρέσεων

Το σχετικό αυτό τμήμα κώδικα έχει ενταχθεί σε ένα block try…catch στο οποίο γίνεται έλεγχος λαθών προκειμένου να αποφευχθεί κάποια ανεπιθύμητη διακοπή του animation κατά την ώρα του runtime. Αν δεν είχε γίνει αυτός ο χειρισμός τότε ήταν πολύ πιθανό να εμφανίζεται το παρακάτω μήνυμα λάθους:

TypeError: Error #1009: Cannot access a property or method of a null object reference.

Το μήνυμα αυτό ουσιαστικά ενημερώνει πως επειδή μερικά αντικείμενα (τύπου Tween) έχουν ακόμα κάποια ανφορά null δεν μπορούν να χρησιμοποιηθούν στο τμήμα κώδικα που ελέγχει την κατάστασή τους. Αυτό είναι προφανές επειδή στο αρχικό τμήμα του κώδικα έχουν δημιουργηθεί κάποιες αναφορές:

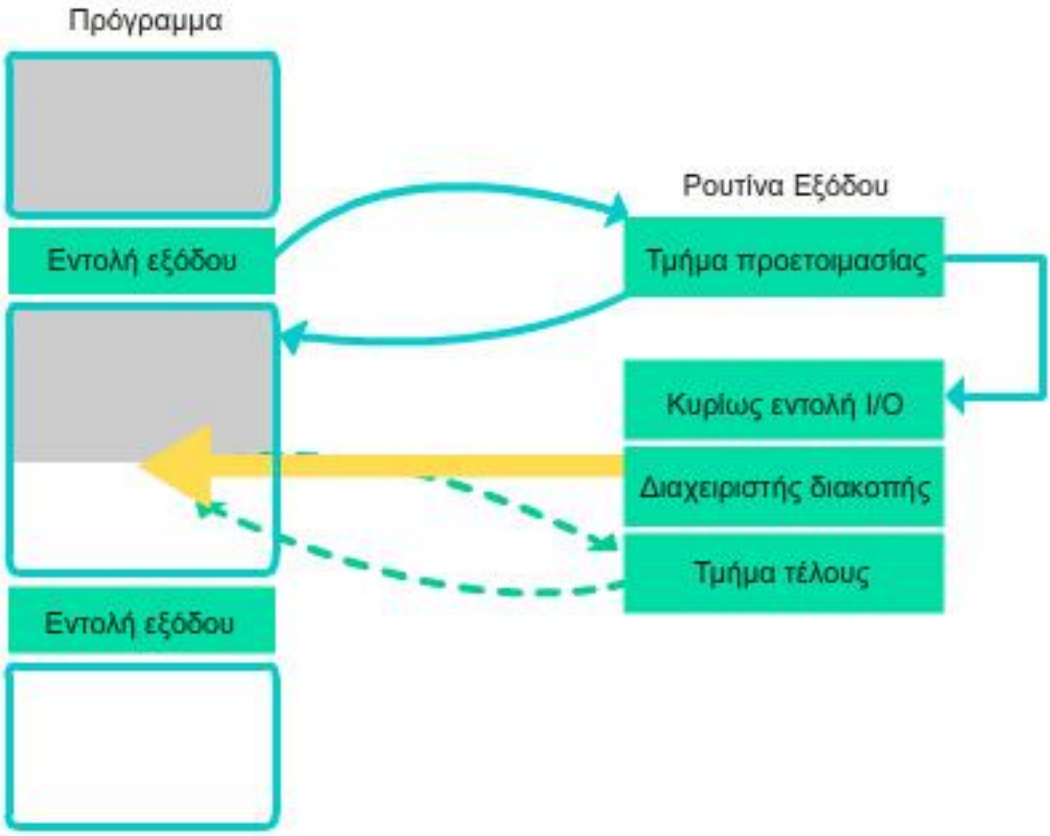
```
var top_panel_tween:Tween;
var middle_panel_tween:Tween;
var bottom_panel_tween:Tween;
```

οι οποίες όμως «γίνονται» αντικείμενα τύπου Tween στη συνέχεια του κώδικα [π.χ. top_panel_tween = new Tween(this.top_panel, "y",None.easeNone,4,67,3,true)].

Επομένως μέχρι να γίνουν αντικείμενα οι αναφορές έχουν την τιμή null και έτσι στο runtime εμφανίζεται το σχετικό μήνυμα λάθους τύπου TypeError.

3.2 Εκτέλεση εντολής εξόδου με διακοπή

Μια παραλλαγή του προηγούμενου σχήματος περιλαμβάνει τον χειρισμό διακοπής (Σχήμα 3.2.1).



Σχήμα 3.2.1: Εκτέλεση εντολής εξόδου με διακοπή

Όπως φαίνεται στην εικόνα, η ΚΜΕ εκτελεί το πρώτο τμήμα της ρουτίνας εξόδου, ειδοποιεί τον ελεγκτή ότι όλα είνατι έτοιμα για την πραγματοποίηση της κυρίως εντολής και συνεχίζει την εκτέλεση των εντολών του κυρίως προγράμματος που βρίσκονται μετά την εντολής εξόδου.

Παράλληλα πραγματοποιείται από την συσκευή I/O η κυρίως εντολή. Όταν αυτή τελειώσει τότε η συσκευή I/O εκδίδει μια αίτηση διακοπής προς την ΚΜΕ (κίτρινο διάνυσμα). Η ΚΜΕ αποδέχεται την αίτηση και ενεργοποιείται ο διαχειριστής της διακοπής ο οποίος μεταφέρει στην ΚΜΕ όσες πληροφορίες χρειάζεται για την ολοκλήρωση της διαδικασίας.

Η ΚΜΕ διακόπτει την λειτουργία της, εκτελεί το τμήμα τέλους, απελευθερώνει την συσκευή και συνεχίζει από το σημείο που δέχθηκε την διακοπή.

Η υλοποίηση της παραπάνω λειτουργίας περιγράφεται στην συνέχεια. Η διαδικασία της υλοποίησης μέχρι και το τμήμα προετοιμασίας είναι ίδια με το προηγούμενο παράδειγμα. Από εκεί και μετά για να επιτευχθεί η παράλληλη εκτέλεση των βημάτων, όπως περιγράφηκε παραπάνω, συντάσεται ο παρακάτω κώδικας:

```
this.vector_to_kme.transform.colorTransform=cYellow;
this.vector_to_IO.transform.colorTransform=cYellow;
myTimerArray[6].addEventListener(TimerEvent.TIMER, panel_02);
myTimerArray[6].start();
myTimerArray[10].addEventListener(TimerEvent.TIMER, kyrios_entoli);
myTimerArray[10].start();
```

Το παραπάνω τμήμα κώδικα περιλαμβάνεται στο βήμα 5 (step05) της συνάρτησης controller ( ) την οποία έχουμε ήδη αναλύσει. Οι δύο πρώτες εντολές αναθέτουν το χρώμα κίτρινο στα ανύσματα vector_to_kme και vector_to_IO τα οποία δείχνουν αντίστοιχα στο $2^{ο}$ τμήμα της ρουτίνας εξόδου και στην κυρίως εντολή I/O.

Στην συνέχεια, με τις επόμενες εντολές χρησιμοποιούμε δύο διαφορετικούς timers οι οποίοι καλούν τις συναρτήσεις panel_02 και kyrios_entoli. Η πρώτη συνάρτηση καλεί εκ νέου τη συνάρτηση controller με όρισμα step08 προκειμένου να συνεχιστεί το $2^{ο}$ τμήμα της ρουτίνας εξόδου, ενώ η δεύτερη συνάρτηση καλεί την συνάρτηση controller με όρισμα step06 προκειμένου να πραγματοποιηθεί η κυρίως εντολή.

Όταν η ροή του animation φτάσει στον διαχειριστή διακοπής (βήμα 7 της συνάρτησης controller()), τότε καλείται η συνάρτηση diakopi. Η τελευταία καλεί την συνάρτηση controller με όρισμα step09. Το βήμα 9 της συνάρτησης controller() περιέχει το παρακάτω τμήμα κώδικα:

```
this.vector_diakopi.visible = true;
```

this.vector_diakopi.transform.colorTransform = cYellow;

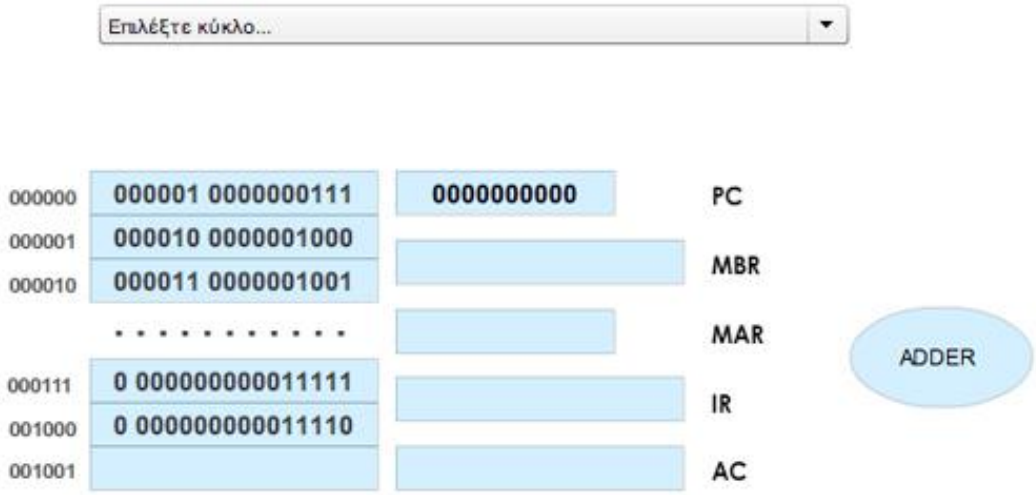
Οι παραπάνω εντολές εμφανίζουν το άνυσμα που περιγράφει την διακοπή και αναθέτουν το χρώμα κίτρινο. Παράλληλα στο βήμα 8 (που αφορά την κίνηση του μεσαίου γκρι πάνελ) μόλις σταματήσει να κινείται (motionFinish) καλείται η συνάρτηση diakopi_02 η οποία καλεί την συνάρτηση controller () με όρισμα step10 όπου αφαιρεί το κίτρινο χρώμα από τον Διαχειριστή διακοπής. Στο ίδιο βήμα υπάρχει και το παρακάτω τμήμα κώδικα:

myTimerArray[12].addEventListener(TimerEvent.TIMER, tmima_tel);
myTimerArray[12].start();

Σκοπός είναι με το πέρας ενός ορισμένου αριθμού δευτερολέπτων να κληθεί η συνάρτηση tmima_tel. όπου κατευθύνει το παράδειγμα στο βήμα 11 της συνάρτησης controller(). Από το βήμα 11 μέχρι και το βήμα 16 υλοποιείται με όμοιο τρόπο το τμήμα τέλους και η εκτέλεση του τμήματος κώδικα της ΚΜΕ από το σημείο που δέχθηκε την διακοπή.

3.3 Κύκλος απόκτησης – εκτέλεσης εντολής σε υποθετικό υπολογιστή

Το επόμενο παράδειγμα αφορά τον Κύκλο απόκτησης - εκτέλεσης εντολής σε υποθετικό υπολογιστή (Σχήμα 3.3.1) υλοποιώντας το πρόγραμμα C=A+B.



Σχήμα 3.3.1: Κύκλος απόκτησης – εκτέλεσης εντολής σε υποθετικό υπολογιστή

Τα βήματα για την απόκτηση – εκτέλεση όλων των εντολών έχουν ομαδοποιηθεί και έχουν καταχωρηθεί σε ένα combo box με όνομα options. Στην αρχική κατάσταση το Combo Box περιέχει το μήνυμα: «Επιλέξτε κύκλο...». Τα πλήκτρα Play | Next και Reset είναι ανενεργά και θα ενεργοποιηθούν με την επιλογή κάποιου βήματος από το Combo Box.

Μόλις επιλεγεί κάποιο βήμα τότε γίνεται και η σχετική αρχικοποίηση (initialization) στην κατάσταση που άφησε το προηγούμενο βήμα. Πιο αναλυτικά, η εντολή:

```
this.options.addEventListener(Event.CHANGE, options_init);
```

σε συνδυασμό με το όρισμα **Event.CHANGE** ανιχνεύει αν έχει προκύψει κάποια αλλαγή στο Combo Box. Επομένως, μόλις επιλεγεί κάποιο από τα οκτώ βήματα καλείται η συνάρτηση options_init() η οποία περιλαμβάνει το ακόλουθο τμήμα κώδικα:

```
function options_init(event:Event):void {
        opt_init();
        this.controllers.znext.mouseEnabled = true;
        this.controllers.zstop.mouseEnabled = true;
}
```

Η παραπάνω συνάρτηση καλεί την συνάρτηση opt_init() η οποία ανιχνεύει ποιο βήμα έχει επιλεγεί από το Combo Box και με μοια συνθήκη του τύπου switch…case καλείται η σχετική συνάρτηση που αρχικοποιεί το βήμα αυτό. Το όνομα της συνάρτησης έχει την μορφή initializeCircleXX() όπου X είναι ο αριθμός του εκάστοτε βήματος. Ως αποτέλεσμα οι συναρτήσεις αρχικοποίησης του κάθε βήματος είναι 7 [από την συνάρτηση initializeCircle02() μέχρι και την συνάρτηση initializeCircle08()].

Με τον ίδιο τρόπο όταν πατηθεί το πλήκτρο zstop το οποίο στο runtime είναι το Reset γίνεται αρχικοποίηση στο κάθε βήμα. Π.χ. Αν κάποιο βήμα βρίσκεται στο $4^{ο}$ στάδιο τότε με το πλήκτρο Reset γίνεται επιστροφή στο $1^{ο}$ στάδιο του ίδιου βήματος.

Κάθε μία συνάρτηση από τις παραπάνω καλεί την συνάρτηση basic_initialize() η οποία περιέχει το τμήμα του κώδικα που απαιτείται για την στοιχειώδη αρχικοποίηση ανεξαρτήτου βήματος. Έτσι με αυτήν την συνάρτηση αποφεύγεται ο πλεονασμός και η επανάληψη ίδιων εντολών σε διαφορετικά σημεία του κώδικα. Αυτές οι εντολές αφορούν την αρχικοποίηση της ιδιότητας alpha στην τιμή 0.2 και της αναίρεσης οποιουδήποτε χρωματισμού που αφορά κάποια θέση μνήμης ή μονάδα. Το τελευταίο επιτυγχάνεται με την γνωστή πλέον χρήση της εντολής:

```
this.[OBJECT].transform.colorTransform = cBlank;
```

Όταν επιλεγεί κάποιο βήμα γίνεται αυτόματα η αρχικοποίησή του και τα πληκτρα Play | Next και Reset ενεργοποιούνται με τις παρακάτω εντολές:

```
this.controllers.znext.mouseEnabled = true;
this.controllers.zstop.mouseEnabled = true;
```

Με το πάτημα του πλήκτρου Play | Next καλείται η συνάρτηση controller() στην οποία με μια συνθήκη τύπου switch…case γίνεται έλεγχος για το ποιο βήμα έχει επιλεγεί από το Combo Box. Επίσης μέσα σε αυτήν την συνάρτηση περιλαμβάνεται η μεταβλητή k που λειτουργεί ως ένας counter – μετρητής που αυξάνει κατά 1 κάθε φορά που επιλέγεται το πλήκτρο Play | Next. Το τμήμα κώδικα στο Σχήμα 3.3.2 δείχνει παραστατικά τον τρόπο λειτουργίας της συνάρτησης controller() αναλύοντας το πώς εκτελούνται διαδοχικά τα στάδια κάθε βήματος.

Αναλυτικότερα, όταν επιλεγεί π.χ. το βήμα 2 «Κύκλος εκτέλεσης της εντολής LOAD A απόκτηση παράγοντα», τότε καλείται η συνάρτηση controller() και στην συνέχεια εκτελείται το παραπάνω τμήμα κώδικα που αφορά το βήμα 2 ως εξής:

Αυξάνει η τιμή της μεταβλητής k κατά 1. Στην συνέχεια με μια συγχωνευμένη συνθήκη τύπου switch…case εκτελείται το κάθε στάδιο που αφορά το βήμα αυτό. Έτσι με την πρώτη φορά που θα κληθεί η συνάρτηση controller η τιμή της μεταβλητής k θα είναι 1 και θα κληθεί συνάρτηση αρχικοποίησης του

βήματος 2 (initializeCircle02). Την δεύτερη φορά η τιμή της μεταβλητής k θα είναι 2 και θα εκτελεστεί το πρώτο στάδιο του βήματος 2 κ.ο.κ. Όταν τελικά η τιμή της μεταβλητής k αποκτήσει την τιμή 22 θα εκτελεστεί το τμήμα κώδικα που υπάρχει στο μπλοκ default όπου μηδενίζει την τιμή της μεταβλητής k. Αυτό συμβαίνει προκειμένου να εκτελεστεί το 3^{ο} βήμα από το 1^{ο} στάδιο.

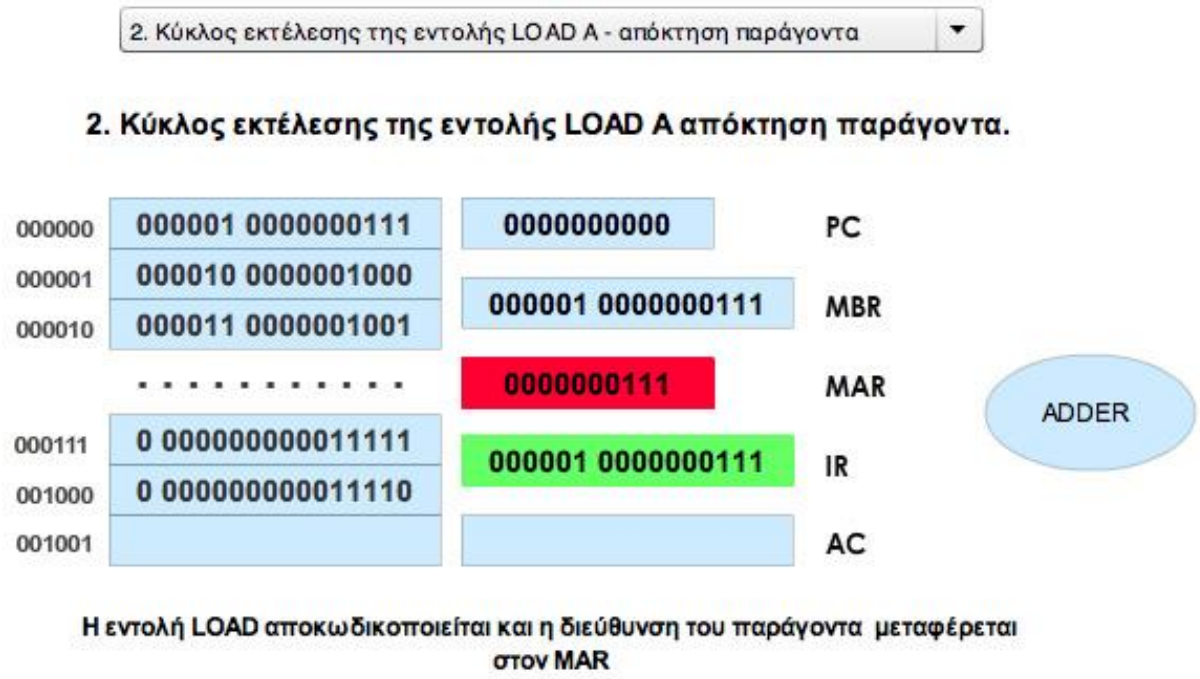
```
case "2":
    k++;
    switch (k){
        case 1:  initializeCircle02(); break;
        case 2:  step02_01(); break;
        case 3:  step02_02(); break;
        case 4:  step03_01(); break;
        case 5:  step03_02(); break;
        case 6:  step04_01(); break;
        case 7:  step04_02(); break;
        case 8:  step04_03(); break;
        case 9:  step05_01(); break;
        case 10: step05_02(); break;
        case 11: step06_01(); break;
        case 12: step06_02(); break;
        case 13: step06_03(); break;
        case 14: step06_04(); break;
        case 15: step07_01(); break;
        case 16: step07_02(); break;
        case 17: step07_03(); break;
        case 18: step08_01(); break;
        case 19: step08_02(); break;
        case 20: step08_03(); break;
        case 21: step08_04(); break;

        default:
            k=0;
        break;
    }

break;
```

Σχήμα 3.3.2: Τμήμα κώδικα συνάρτησης controller()

Η υλοποίηση των επιμέρους σταδίων του κάθε βήματος περιλαμβάνει τον σχετικό χρωματισμό των θέσεων μνήμης ή των αντίστοιχων μονάδων καθώς και τα λεκτικά που περιγράφουν το τι επιτελείται σε κάθε στάδιο του βήματος (Σχήμα 3.3.3).



Σχήμα 3.3.3: Στιγμιότυπο ενός βήματος

Η παραπάνω εικόνα περιγράφει το 1ο στάδιο του 2ου βήματος το οποίο εκτελείται από το τμήμα κώδικα της συνάρτησης step02_01() (Σχήμα 3.3.4).

```
function step02_01():void {
    this.ir_text_cont.transform.colorTransform=cGreen;
    this.ir_text_cont.alpha = 0.2;
    this.ir_text_cont.alpha = 1;
    this.ir.text=this.thesi01.text;
    this.controllers.znext.mouseEnabled = false;

    this.mbr_text_cont.transform.colorTransform=cBlank;
    this.mbr_text_cont.alpha =0.2;

    var myTimer4:Timer = new Timer(1000, 1);
        myTimer4.addEventListener(TimerEvent.TIMER, timerHandler4);
        myTimer4.start();

    this.messages.text="Η εντολή LOAD αποκωδικοποιείται και η διεύθυνση του ...
}
```

Σχήμα 3.3.4: Η συνάρτηση step02_01()

Αρχικά το panel που αναπαριστά την μονάδα IR (Instruction Register) πέρνει το χρώμα πράσινο με την τιμή 1 στην ιδιότητα alpha. Στην συνέχεια το κείμενο του IR πέρνει την τιμή της θέσης 1 της μνήμης σύμφωνα με την θεωρία του παραδείγματος.

Κατόπιν το panel (mbr_text_cont) που αναπαριστά την μονάδα MBR επανέρχεται στην αρχική κατάσταση (ανοικτό γαλάζιο).

Επιπλέον. δημιουργείται ένα αντικείμενο τύπου Timer με όνομα myTimer4 και χρόνο 1 δευτερόλεπτο ή 1000 mseconds. Με το πέρας του ενός δευτερολέπτου καλείται η συνάρτηση timerHandler4(). Τέλος, ενημερώνεται και το κείμενο με το σχετικό λεκτικό.

Η συνάρτηση timerHandler4() εκτελεί την συνέχεια του 1^{ου} σταδίου του 2^{ου} βήματος το οποίο περιλαμβάνει τα εξής:

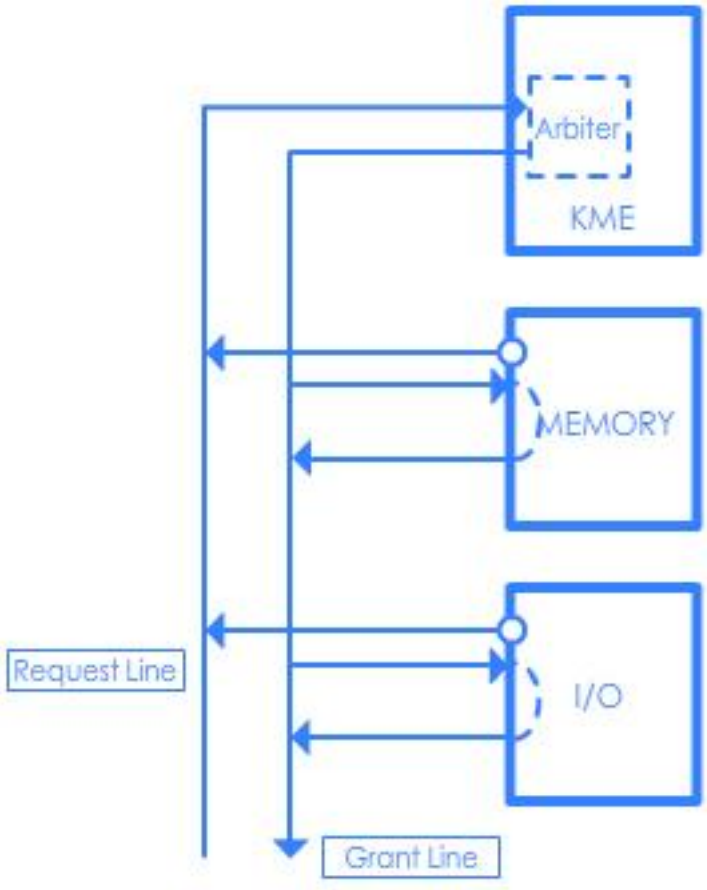
Το panel που αναπαριστά την μονάδα MAR αποκτά χρώμα κόκκινο με τιμή 1 στο alpha, κατόπιν ενημερώνεται και το κείμενο του MAR με τα σχετικά ψηφία και τέλος ενεργοποιείται ξανά το πλήκτρο znext ώστε όταν πατηθεί να προχωρήσει στο επόμενο στάδιο.

Με όμοιο τρόπο εκτελούνται και τα επόμενα βήματα με τα επιμέρους στάδια.

3.4 Κεντρική Διαιτησία του πλησιέστερου προς την ΚΜΕ

Στο επόμενο παράδειγμα που αφορά την Κεντρική Διαιτησία του πλησιέστερου προς την ΚΜΕ (Σχήμα 3.4.1), θα επισημανθούν και θα αναπτυχθούν τα παρακάτω χαρακτηριστικά:

- Motion Guide Layers
- Global Variables



Σχήμα 3.4.1: Κεντρική Διαιτησία του πλησιέστερου προς την KME

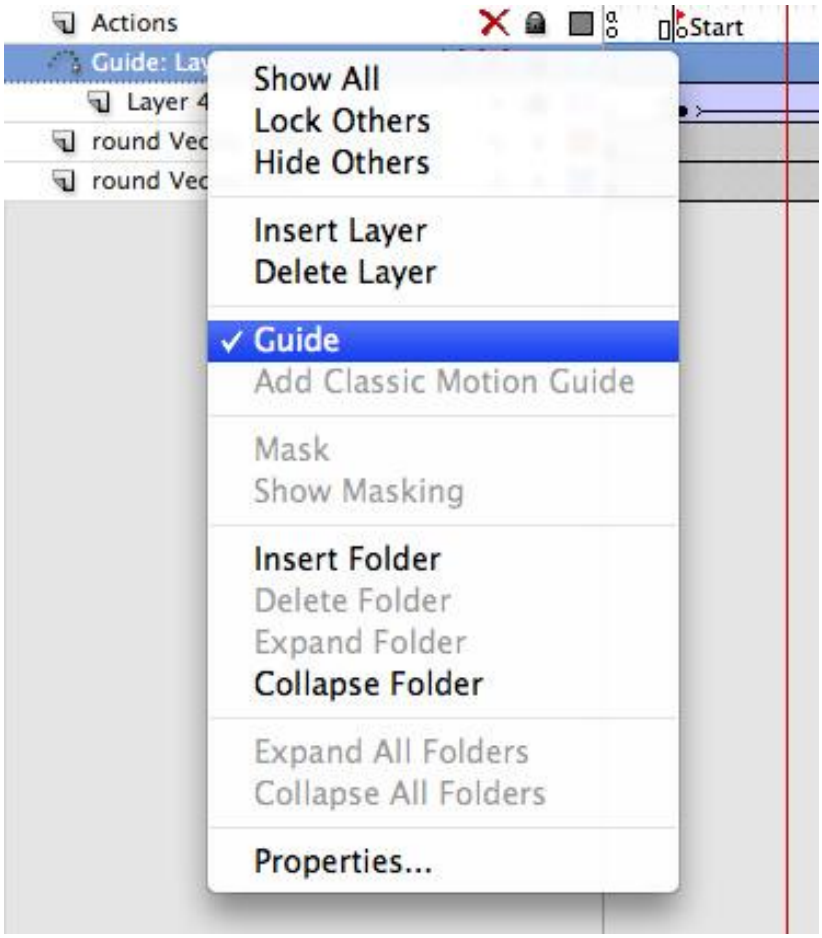
Αρχικά έχουν δημιουργηθεί τα vectors που αφορούν τις Request Line και Grant Line. Επίσης, έχουν δημιουργηθεί τρία object που αντιπροσωπεύουν τις μονάδες KME, MEMORY και I/O. Επισημαίνεται πως κάθε object δεν έχει φόντο αλλά περιλαμβάνει ένα τετράγωνο περίβλημα και ένα text που αναφέρει το όνομα της κάθε μονάδας (π.χ. MEMORY). Ο λόγος αφορά τιν ιδιότητα tint που θα χρωματίζει με πράσινο την μονάδα MEMORY ή την I/O όταν αυτή θα δέχεται το σήμα (Σχήμα 3.4.2)

Σχήμα 3.4.2: Αποτέλεσμα ιδιότητας Tint

3.4.1 Motion Guide Layers

Στην συνέχεια θα αναλύσουμε μια νέα τεχνική που αφορά την χρήση των **Motion guide layers**. Τα motion guide layers δίνουν την δυνατότητα σχεδιασμού path (μονοπάτι) πάνω στο οποίο θα κινηθούν τα instances ή οποιοδήποτε άλλο object θέλουμε να κινηθεί πάνω στην τροχιά του Guide Layer. Μπορούν να συνδεθούν παραπάνω από ένα layers σε ένα motion guide layer και όλα μαζί θα ακολουθούν την ίδια τροχιά. Για την δημιουργία ενός motion guide layer η διαδικασία είναι η εξής:

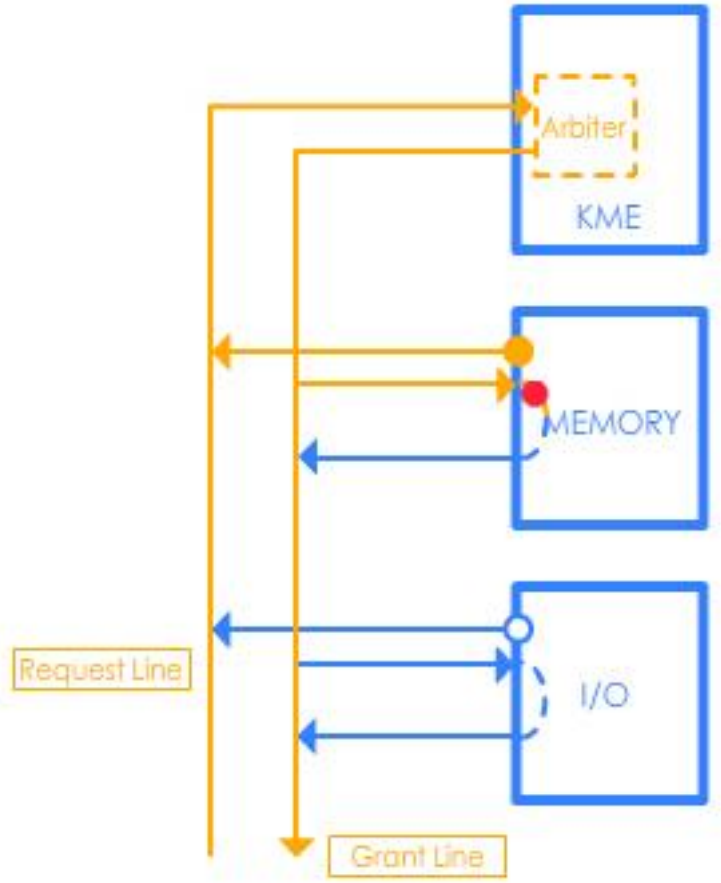
Έχει σχεδιαστεί η τροχιά (path) σε ένα απλό layer. Στην συνέχεια κάνουμε δεξί κλικ πάνω στο layer και επιλέγουμε Guide (Σχήμα 3.4.3).



Σχήμα 3.4.3: Μετατροπή ενός layer σε guide layer

Τότε το layer μετατρέπεται σε Guided Layer. Έπειτα με drag-n-drop συνδέουμε τα layers των οποίων τα objects θέλουν να κινηθούν σύμφωνα με την τροχιά του Guided Layer.

Στην υλοποίηση του παραδείγματος, έχει δημιουργηθεί ένα guided layer στο οποίο έχει σχεδιαστεί το μονοπάτι που θα ακολουθήσει το σήμα στην Grant Line (Σχήμα 3.4.4).



Σχήμα 3.4.4: Στιγμιότυπο πορείας σήματος

3.4.2 Global Variables

Για την συνέχεια της περιγραφής της υλοποίησης του παραδείγματος, χρειάζεται να επεξηγηθεί οι έννοια και ο σκοπός ύπαρξης των **Global Variables**.

Οι μεταβλητές global είναι αυτές που μπορούν να χρησιμοποιηθούν σε όλα τα τμήματα του κώδικα. Η δημιουργία γίνεται εκτός κλάσης ή συνάρτησης. Μέχρι και την ActionScript 2.0 η κλήση των μεταβλητών αυτών μπορούσε να γίνει και από τα timelines των επιμέρους instances, movieclip κοκ. Στην ActionScript 3.0 όμως, δεν είναι δυνατός αυτός ο τρόπος. Επειδή όμως στην υλοποίηση του παραδείγματος θα χρειαστεί να χρησιμοποιηθούν κάποιες variable μεταβλητές από διαφορετικά σημεία του κώδικα και από τα layers που βρίσκονται μέσα στα movieclips, αναπτύχθηκε ο παρακάτω τρόπος.

Δημιουργήθηκε ένα ξεχωριστό αρχείο με όνομα **Globals.as**. Τα αρχεία με κατάληξη ***.as** είναι αρχεία τα οποία περιλαμβάνουν μόνο κώδικα ActionScript, περιέχουν πακέτα και κλάσεις των οποίων τα αντικείμενα και οι μεταβλητές μπορούν να προσπελαστούν από οποιοδήποτε εξωτερικό αρχείο με κατάληξη *.swf ή *.as. Ο πλήρης κώδικας αναφέρεται στο Παράρτημα Α’ και περιλαμβάνει την δημιουργία των αντικειμένων τύπου Color (cGreen, cYellow, cBlank), τις αναφορές των SimpleButton και των MovieClip καθώς και τον πίνακα που περιλαμβάνει αντικείμενα τύπου Timer.

Επομένως, έχοντας αναλύσει και τα παραπάνω, η υλοποίηση του παραδείγματος που αφορά την Κεντρική Διαιτησία του πλησιέστερου προς την ΚΜΕ έχει ως εξής:

Αφού έχει δημιουργηθεί ένα Layer με όνομα Actions στο πρώτο keyframe γράφουμε τον κώδικα που θα αποτελέσει τον πυρήνα του προγράμματος. Στις πρώτες σειρές γίνεται η εισαγωγή των κλάσεων Timer, TimerEvent, Sprite, Color και των κλάσεων του πακέτου display. Αμέσως μετά με την εντολή -import Globals-γίνεται η εισαγωγή του πακέτου globals που βρίσκεται στο αρχείο Globals.as.

Στο Σχήμα 3.4.5 φαίνεται το τμήμα κώδικα που αντιστοιχεί τις Global αναφορές αντικειμένων με τα αντίστοιχα αντικείμενα που βρίσκονται μέσα στο πρόγραμμα.

```
Globals.GmemoryBTN          = this.memoryBTN;
Globals.GioBTN              = this.IObtn;
Globals.GrequestLine        = this.requestLine;
Globals.GgrantLine          = this.grantLine;
Globals.GvectorMemory2RLine = this.vectorMemory2RLine;
Globals.GvectorIO2RLine     = this.vectorIO2RLine;
Globals.Gdescriptions       = this.descriptions;
Globals.Gzreset             = this.controllers.zstop;
```

Σχήμα 3.4.5 Αντιστοίχιση Global με των Local μεταβλητών

Με αυτόν τον τρόπο οι αναφορές δείχνουν στα αντίστοιχα αντικείμενα.

Η εκκίνηση του animation γίνεται με την επιλογή (αριστερό κλικ) κάποιας μονάδας (MEMORY ή I/O). Το τμήμα κώδικα που ανιχνεύει το αριστερό κλικ είναι ο παρακάτω:

```
Globals.GmemoryBTN.addEventListener(MouseEvent.CLICK,eventHandler);
Globals.GioBTN.addEventListener(MouseEvent.CLICK,eventHandler);
```

Το πρώτο χαρακτηριστικό που παρατηρείται είναι ότι η προσπέλαση των πλήκτρων που περιγράφουν την MEMORY και την I/O γίνεται με την χρήση των αντικειμένων Globals.GmemoryBTN και Globals.GioBTN αντίστοιχα.

3.4.3 Χειρισμός των events της ροής του προγράμματος

Η συνάρτηση eventHandler χειρίζεται τα events που προκύπτουν από το πάτημα της εκάστοτε μονάδας. Όπως έχει επεξηγηθεί σε προηγούμενο παράδειγμα με τον κώδικα event.target.name αποκτάται το όνομα του αντικειμένου που προκάλεσε το event. Η τιμή αυτή αποθηκεύεται σε μια Global μεταβλητή την unit.

Στην συνέχεια με μία συνθήκη του τύπου if…else εκτελείται και ο σχετικός κώδικας ο οποίος περιλαμβάνει την εμφάνιση του κατάλληλου λεκτικού που περιγράφει την τρέχουσα κατάσταση του παραδείγματος καθώς επίσης και την εμφάνισει με κίτρινο χρώμα του μηνύματος sendingRequest. Μετά την συνθήκη if…else καλείται η συνάρτηση toRequestLine(event) (Σχήμα 3.4.6).

```
function toRequestLine(event:Event):void
{
    if(Globals.unit=="memoryBTN")
    {
        Globals.GmemoryBTN.transform.colorTransform          = Globals.cBlank;
        Globals.GvectorMemory2RLine.transform.colorTransform = Globals.cYellow;
    }
    else
    {
        Globals.GioBTN.transform.colorTransform              = Globals.cBlank;
        Globals.GvectorIO2RLine.transform.colorTransform     = Globals.cYellow;
    }
        Globals.myTimerArray[2].addEventListener(TimerEvent.TIMER, rLine);
        Globals.myTimerArray[2].start();
}
```

Σχήμα 3.4.6: Η συνάρτηση toRequestLine()

Η συνάρτηση toRequestLine έχει σκοπό να αρχίσει το animation ανάλογα με την μονάδα που το κάλεσε. Εξετάζει την global μεταβλητή unit προκειμένου να διαπιστώσει ποιο πλήκτρο (MEMORY ή I/O) πατήθηκε και κατόπιν με μια συνθήκη if…else χρωματίζει με κίτρινο το άνυσμα που στέλνει σήμα από την εκάστοτε μονάδα.

Στην συνέχεια με τον κατάλληλο χρονισμό συνεχίζει στην συνάρτηση rLine() η οποία υλοποιεί την συνέχεια του animation χρωματίζοντας με κίτρινο χρώμα την Request Line καλώντας με την σειρά της ύστερα από πεπερασμένο χρονικό διάστημα την συνάρτηση arbiter().

Η συνάρτηση arbiter() χρωματίζει με κίτρινο χρώμα την υπομονάδα arbiter (διαιτητής), που βρίσκεται εντός της ΚΜΕ, όταν φτάσει σε αυτήν το σήμα από την Request Line. Στην συνέχεια προωθεί το μήνυμα στην Grant Line. Αυτό επιτυγχάνεται με το παρακάτω τμήμα κώδικα:

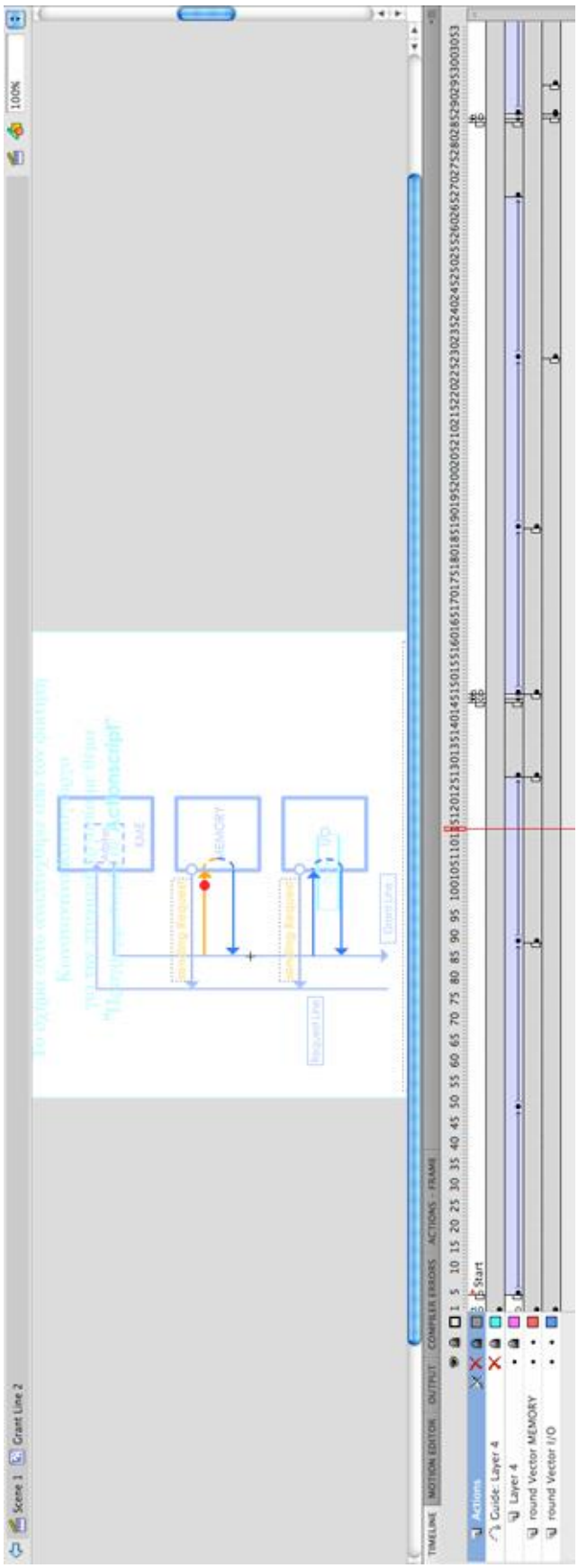
```
Globals.myTimerArray[4].addEventListener(TimerEvent.TIMER,toGrantLine);
Globals.myTimerArray[4].start();
```

Το παραπάνω τμήμα κώδικα ύστερα από συγκεκριμένο χρονικό διάστημα καλεί την συνάρτηση GrantLine().

Η συνάρτηση GrantLine() χρωματίζει με κίτρινο χρώμα την Grant Line προκειμένου να φανεί ότι είναι ενεργή και πως θα μεταφέρει το σήμα προς την MEMORY ή την I/O. Έπειτα το αμέσως επόμενο τμήμα κώδικα:

```
if (this.grantLineSignals.currentFrame == 1 ||
   this.grantLineSignals.currentFrame == 149 ||
   this.grantLineSignals.currentFrame == 289)
        {
              this.grantLineSignals.gotoAndPlay(5);
        }
```

ελέγχει αν το playback του movieClip με όνομα grantLineSignals (Σχήμα 3.4.7) βρίσκεται στο frame 1 ή στο 149 ή στο 289. Σε αυτήν την περίπτωση θα παίξει από το frame 5.



Σχήμα 3.4.7: Το timeline της πορείας του σήματος

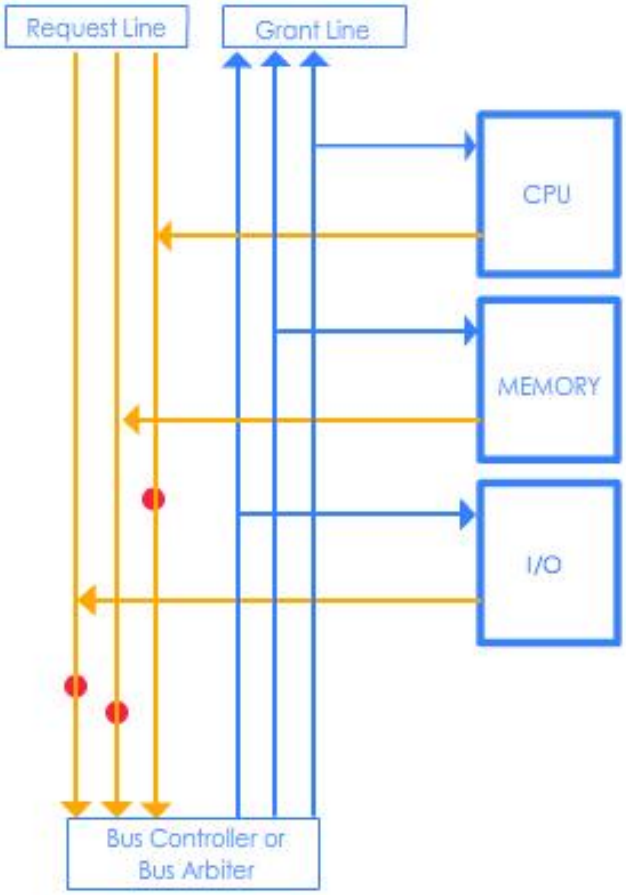
Σε αυτό το MovieClip υλοποιείται το animation που αφορά την τροχιά που ακολουθεί το σήμα, το οποίο είναι σχεδιασμένο ως μια κόκκινη σφαίρα. Έχει εφαρμοστεί η τεχνική του Guided Layer στο οποίο υπάρχει το path που συμπίπτει με την grant Line.

Στα keyframe 148,149 και 150 του timeline του MovieClip grantLineSignals γίνονται οι σχετικοί έλεγχοι που αφορούν αν το σήμα προορίζεται προς την MEMORY ή όχι. Στην πρώτη περίπτωση η MEMORY αποδέχεται το σήμα και πρασινίζει (αναπαριστώντας ότι το αποδέχθηκε) η στέλνει το σήμα στην Grant Line προκειμένου να ακολουθήσει την πορεία μέχρι την I/O στα keyframe 288 και 289. Η I/O ελέγχει αν το σήμα αφορά την ίδια και το αποδέχεται ή όχι.

Το ιδιαίτερο χαρακτηριστικό του παραπάνω παραδείγματος είναι ο συνδυασμός ελέγχου της σειριακής κίνησης ενός εσωτερικού timeline κάποιου αντικειμένου Movie Clip με τη χρήση κώδικα σε ένα μόνο keyframe στο κεντρικό timeline.

3.5 Κεντρική Διαιτησία με προτεραιότητες

Σε αυτό το παράδειγμα υλοποιείται ένα άλλος κεντρικός μηχανισμός διαιτησίας, περισσότερο πολύπλοκος, με καθορισμό προτεραιοτήτων. Απαιτούνται τόσα ζεύγη γραμμών ελέγχου αίτησης – χορήγησης (request – grant) όσες και οι μονάδες (Σχήμα 3.5.1).



Σχήμα 3.5.1: Κεντρική Διαιτησία με προτεραιότητες

Βλέπουμε πως ο διαιτητής δέχεται αιτήσεις από τις μονάδες μέσω των ανεξαρτήτων γραμμών αίτησης και αποφασίζει με βάση έναν πίνακα προτεραιοτήτων το οποίο ορίζει ο κατασκευαστής. Στο συγκεκριμένο παράδειγμα η σειρά προτεραιοτήτων είναι: CPU, MEMORY, I/O.

Όταν φτάνουν ταυτόχρονα πολλές αιτήσεις (requests) από πολλές μονάδες στον Bus Controller ή Bus Arbiter τότε θα σταλούν τα σήματα στην grant line με βάση την ιεραρχία, ανεξάρτητα από το ποιο σήμα έφτασε πρώτο στον Controller. π.χ. Αν στείλουν αιτήματα η I/O και η CPU τότε στην grant Line θα σταλεί τον διάδρομο θα αναλάβει πρώτα η CPU και μετά η I/O.

Όσον αφορά την υλοποίηση, έχουν σχεδιαστεί τα κατάλληλα γραφικά που αφορούν τις grant line, request line, τις μονάδες και τον bus controller.

Τα γραφικά που αφορούν τις μονάδες CPU, MEMORY και I/O έχουν μετασχηματιστεί στα αντίστοιχα buttons: CPUbtn, MEMORYbtn και IObtn τα οποία όταν πατηθούν καλούν τις συναρτήσεις CPU_Request, MEMORY_Request και IO_Request, σύμφωνα με το ακόλουθο τμήμα κώδικα:

```
IObtn.addEventListener(MouseEvent.MOUSE_DOWN, IO_Request);
CPUbtn.addEventListener(MouseEvent.MOUSE_DOWN, CPU_Request);
MEMORYbtn.addEventListener(MouseEvent.MOUSE_DOWN,
MEMORY_Request);
```

Η συνάρτηση CPU_Request έχει σκοπό να ενεργοποιήσει την διαδικασία request και χρωματίζει με κίτρινο το άνυσμα από την CPU προς την Request Line που αφορά αποκλειστικά την CPU. Προηγουμένως έχει δημιουργηθεί ένα object με όνομα cpu_signal το οποίο είναι ένας κόκκινος κύκλος που αναπαριστά το σήμα που μεταφέρεται από και προς τις σχετικές μονάδες.

Το επόμενο τμήμα κώδικα:

```
cpu_signal.y = 207;
CPU_SignalRequestA =
        new Tween(cpu_signal, "x", None.easeNone, 215, 69, 2, true);
CPU_signalRequest02 = setInterval(CPU_Request01, 2000);
```

τοποθετεί το cpu_signal στην θέση 207 του άξονα y και κατόπιν δημιουργεί μια κίνηση Tween στον άξονα x από την θέση 215 στην 69 διάρκειας 2 δευτερολέπτων. Σε προηγούμενη ενότητα έχει αναλυθεί διεξοδικά η λειτουργία των αντικειμένων τύπου Tween.

Αξίζει να αναφερθεί πως μια εναλλακτική συνάρτηση που λειτουργεί ως χρονισμός είναι η **setInterval().** Δέχεται δύο ορίσματα: το όνομα της επόμενης συνάρτησης που θα κληθεί μετά το πέρας συγκεκριμένου αριθμού δευτερολέπτων και των αριθμό των δευτερολέπτων με λεπτομέρεια msecs. Έτσι, η παραπάνω εντολή θα καλέσει την συνάρτηση CPU_Request01 μετά από 2 δευτερόλεπτα.

Η συνάρτηση CPU_Request01() ενεργοποιεί την Request Line χρωματίζοντάς την με κίτρινο. Τοποθετεί το cpu_signal στην θέση 69 του άξονα x και με ένα αντικείμενο τύπου Tween υλοποιεί την κίνηση του σήματος από την θέση 207 στην 472 του άξονα y που καταλήγει ουσιαστικά στον Bus Controller, ενώ με τον τερματισμό της κίνησης καλείται η συνάρτηση CPU_Boolean. Η συνάρτηση CPU_Boolean αναθέτει στην Boolean μεταβλητή CPU την τιμή true, καλεί την συνάρτηση Bus_Controller() και τέλος αποκαθιστά τον χρωματισμό της request line που αφορά την CPU.

Με όμοιο τρόπο έχουν υλοποιηθεί και οι σχετικές μέθοδοι, μέχρι και την κλήση της Bus_Controller(), των μονάδων MEMORY και I/O.

Η συνάρτηση Bus_Controlller() ενεργοποιεί τον Bus Controller χρωματίζοντας με κίτρινο τον Controller. Στην συνέχεια καλεί την συνάρτηση Bus_Controller_Distrib().

Η συνάρτηση Bus_Controller_Distrib() αποτελείται από τον παρακάτω κώδικα (Σχήμα 3.5.2)

```
function Bus_Controller_Distrib(event:Event):void
{
   unitGrantLinesReset();
   if(CPU==true)
   {
      this.lines.bus_controller.transform.colorTransform = cYellow;
      CPU_Grant01();
   }
   else if(CPU == false && MEMORY==true)
   {
      this.lines.bus_controller.transform.colorTransform = cYellow;
      MEMORY_Grant01();
   }
   else if(CPU == false && IO == true)
   {
      this.lines.bus_controller.transform.colorTransform = cYellow;
      IO_Grant01();
   }
}
```

Σχήμα 3.5.2: Η συνάρτηση Bus_Controller_Distrib()

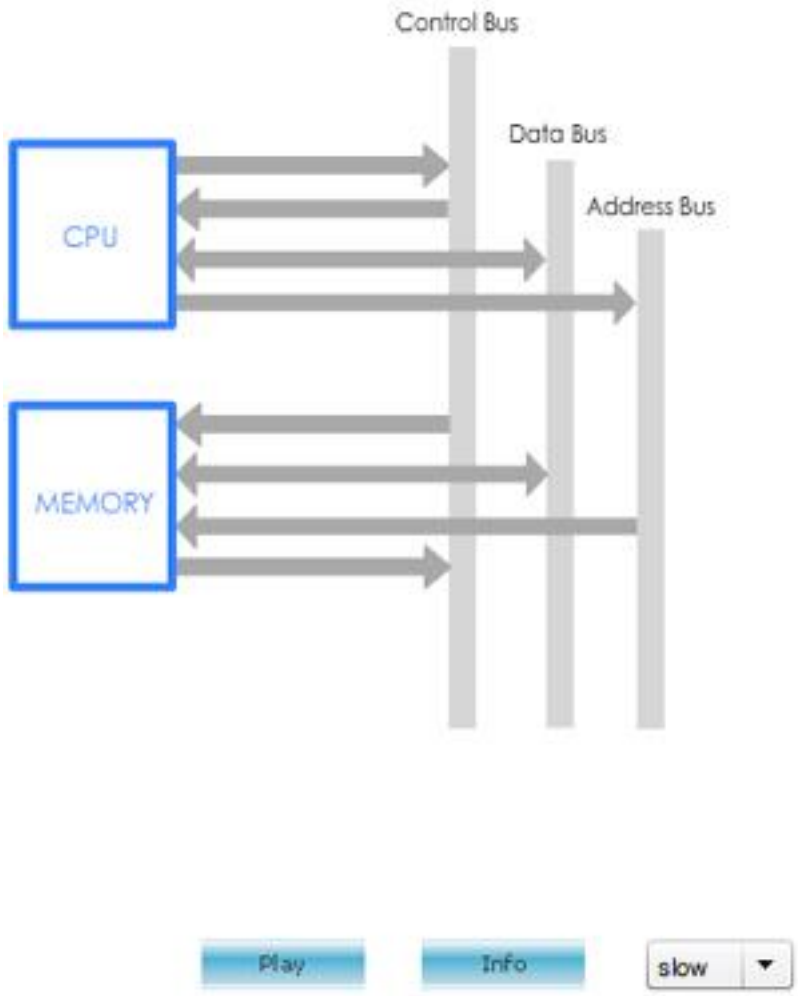
Ο κώδικας υλοποιεί την ανάθεση στην grant line του σήματος που αφορά την κάθε μονάδα με βάση την ιεραρχία που περιγράφηκε νωρίτερα. Έτσι:

- Αν η CPU έχει κάνει request τότε ενεργοποιείται άμεσα η grant line της CPU και στέλνεται το σήμα ανεξάρτητα από το αν συνυπάρχει αίτηση από άλλη μονάδα
- Αν η CPU δεν έχει κάνει request αλά έχει κάνει η MEMORY τότε προωθείται το σήμα που αφορά την MEMORY ανεξάρτητα από το αν έχει κάνει αίτηση η I/O.
- Αν η CPU και η MEMORY δεν έχουν κάνει request αλλά έχει κάνει η I/O τότε προωθείται το σήμα της I/O.

Με όμοιο τρόπο εκτελούνται οι συναρτήσεις που αφορούν την μετάβαση του σήματος στην εκάστοτε μονάδα μέσω της σχετικής τους Grant Line. Στην συνέχεια, όταν το σήμα φτάσει στον προορισμό του (CPU, MEMORY ή I/O) τότε καλείται εκ νέου η συνάρτηση Bus_Controller_Distrib() για να γίνει η μετάβαση του σήματος της επόμενης μονάδας (αν κάποια μονάδα έχει κάνει request), ενώ παράλληλα η Boolean μεταβλητή (CPU, MEMORY ή I/O) παίρνει την τιμή false.

3.6 Εγγραφή ΚΜΕ στον δίαυλο

Εξετάζεται το παράδειγμα όπου μόνο ένα ζεύγος συσκευών μπορεί να κατέχει τον δίαυλο. Το ζέυγος συσκευών είναι η CPU (ΚΜΕ) και η MEMORY, ενώ οι δίαυλοι επικοινωνίας είναι ο Control Bus, ο Data Bus και ο Address Bus (Σχήμα 3.6.1).



Σχήμα 3.6.1: Εγγραφή ΚΜΕ στον δίαυλο

Επιπρόσθετα έχει προστεθεί ένα Combo Box που περιέχει δύο επιλογές: Slow, Fast. Οι επιλογές σχετίζονται με την ταχύτητα που επιθυμεί ο χρήστης να εκτελείται το animation.

Το animation αρχίζει όταν ο χρήστης επιλέγει την επιθυμητή ταχύτητα και κάνει αριστερό κλικ στην μονάδα CPU ή στο πλήκτρο Play (στιγμιότυπο znext). Τότε καλείται η συνάρτηση eventHandler() η οποία με την σειρά της καλεί την συνάρτηση controller() με όρισμα step01. Η συνάρτηση διαχειρίζεται όλα τα βήματα του παραδείγματος με μία συνθήκη τύπου switch..case, όπου παρόμοια δομή έχει χρησιμοποιηθεί και σε προηγούμενα παραδείγματα. Στην περίπτωση του πρώτου βήματος (case “step01”), τότε καλείται αμέσως η συνάρτηση start_cpu() και στην συνέχεια ύστερα από συγκεκριμένο χρονικό διάστημα μεταβαίνει στο 2ο βήμα όπως φαίνεται στο παρακάτω τμήμα κώδικα:

```
myTimerArray[1].addEventListener(TimerEvent.TIMER, to_step02);
myTimerArray[1].start();
```

Η συνάρτηση start_cpu() αποτελείται από το παρακάτω τμήμα κώδικα (Σχήμα 3.6.2):

```
function start_cpu():void {
    this.cpu.mouseEnabled=false;
    this.znext.mouseEnabled=false;
    this.cpu.transform.colorTransform= cGreen;

    if (this.speed_option.value=="slow") {
        speed=5000;
    } else {
        speed=1500;
    }

    for (var i:int=1; i<11; i++) {
        var myTimer:Timer=new Timer(speed,1);
        myTimerArray[i]=myTimer;
    }
}
```

Σχήμα 3.6.2: Η συνάρτηση start_cpu()

Απενεργοποιεί τα button objects (cpu και znext), ενεργοποιεί την CPU, χρωματίζοντάς την με πράσινο. Το επόμενο τμήμα διαμορφώνει την ταχύτητα εκτέλεσης του animation σύμφωνα με την προτίμηση του χρήστη από το αντίστοιχο Combo Box. Η πρώτη σειρά ελέγχει αν η επιλογή του Combo Box με όνομα speed_option είναι «slow». Σε αυτήν την περίπτωση, ανατίθεται στην μεταβλητή speed η τιμή 5000 διαφορετικά θα πάρει την τιμή 1000. Αμέσως μετά με το loop δημιουργείται πίνακας αντικειμένων τύπου Timer με όρισμα χρονικής διάρκειας την τιμή της μεταβλητής speed. Οι τιμές 5000 ή 1000 της μεταβλητής speed σημαίνουν (5000 ή 1000) mseconds ή (5 ή1) δευτερόλεπτα.

Στο 2^{ο} βήμα οι δίαυλοι Control Bus και Address Bus χρωματίζονται με χρώμα μπλε και κόκκινο αντίστοιχα. Επίσης εμφανίζονται τα μηνύματα MREQ-RD

και Address στα δυναμικά TextFields προκειμένου να περιγράψουν το τί περιέχουν τα σήματα που μεταφέρονται στους αντίστοιχους διαύλους.

Σειριακά εκτελούνται και τα υπόλοιπα βήματα μέχρι και την στιγμή που φτάνουν στην MEMORY. Στην συνέχεια στο βήμα 4 καλείται ταυτόχρονα και η συνάρτηση memoryPulses() η οποία περιλαμβάνει το τμήμα κώδικα της παρακάτω εικόνας (Σχήμα 3.6.3).

```
function memoryPulses():void {
    s++;
    if (this.speed_option.value=="fast") {scond=16;}
    else {scond=40;}

    if (s<scond) {
        var myModulo:Number=s%2;
        if (myModulo==1) {
            memoryPulse.addEventListener(TimerEvent.TIMER, to_MagentaLight);
            memoryPulse.start();
        } else if (myModulo==0) {
            memoryPulse2.addEventListener(TimerEvent.TIMER, to_Magenta);
            memoryPulse2.start();
        }
    } else
        {
            s=0;
            this.memory.transform.colorTransform = cBlank;
        }

}
```

Σχήμα 3.6.3: Η συνάρτηση memoryPulses()

Η παραπάνω συνάρτηση υλοποιεί τους παλμούς που αφορούν την πραγματοποίηση του κύκλου κύκλο μνήμης. Στο animation η μνήμη περιγράφει τον κύκλο με την εναλλαγή των χρωμάτων Magenta Light και Magenta και αυτό επιτυγχάνεται με την παραπάνω συνάρτηση. Αναλυτικότερα, έχει δημιουργηθεί η μεταβλητή s που λειτουργεί ως counter. Με την επόμενη σειρά κώδικα ελέγχεται αν η επιλογή του Combo Box που αφορά την ταχύτητα εκτέλεσης του animation είναι fast ή slow. Στην πρώτη περίπτωση μια άλλη μεταβλητή, που ορίζει τον αριθμό των παλμών, παίρνει την τιμή 16 και βάση της αναλογίας στην δεύτερη περίπτωση παίρνει την τιμή 40. Στην συνέχεια με την εντολή: var myModulo:Number = s%2 δημιουργείται η μεταβλητή myModulo στην οποία

ανατίθεται το υπόλοιπο της διαίρεσης της μεταβλητής – counter s με τον αριθμό 2 το οποίο είναι 0 ή 1. Επομένως είναι πλέον έυκολο να κατανοηθεί ότι οι επόμενες if συνθήκες προσδίδουν το χρώμα Magenta Light ή Magenta εναλλάξ ανάλογα με το ακέραιο υπόλοιπο.

Έτσι, το αποτέλεσμα των παραπάνω είναι αυτό της παρακάτω εικόνας (Σχήμα 3.6.4):

Σχήμα 3.6.4: Στιγμιότυπα των παλμών της MEMORY

Στην συνέχεια, η MEMORY στέλνει τα σχετικά σήματα στον δίαυλο Data Bus και στον Data Ready.

Τέλος η CPU κάνει αποδοχή των σημάτων και μετά το πέρας συγκεκριμένου χρονικού διαστήματος γίνεται αρχικοποίηση.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- John L. Hennessy and David A. Patterson, "Computer Organization and Design : The Hardware/Software Interface", Academic Press, 1997.
- David A. Patterson, John L. Hennessy, and David Goldberg, "Computer Architecture : A Quantitative Approach", 3th ed., Academic Press, 2003.
- John L. Hennessy and David A. Patterson, “Αρχιτεκτονική Υπολογιστών”, Εκδόσεις Τζιόλα 2008
- William Stallings, "Computer Organization & Architecture", Prentice Hall, 2003
- http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/ 3/12/2009
- http://livedocs.adobe.com/flash/9.0/main/flash_as2_learning.pdf Μάιος, 2009

ΠΑΡΑΡΤΗΜΑ Α'

Κώδικας πακέτου Globals:

```
package
{
	public class Globals
	{
		import flash.utils.Timer;
		import flash.events.TimerEvent;
		import flash.display.Sprite;
		import fl.motion.Color;
		import flash.display.*;
		import flash.text.TextField;

		public static var GmemoryBTN:SimpleButton;
		public static var GioBTN:SimpleButton;
		public static var GrequestLine:MovieClip;
		public static var GgrantLine:MovieClip;
		public static var GvectorMemory2RLine:MovieClip;
		public static var GvectorIO2RLine:MovieClip;
		public static var Gdescriptions:TextField;
		public static var Gzreset:SimpleButton;

		public static var unit:String;
		public static var cGreen:Color=new Color();
				cGreen.setTint(0x65ff65, 1);
		public static var cYellow:Color=new Color();
```

```
                cYellow.setTint(0xff9900, 1);

        public static var cBlank:Color=new Color();

                cBlank.setTint(0xffffff, 0);

        public static var i:int;

        public static var temp:int=0;

        public static var slave:Boolean=false;



        public static var myTimerArray:Array = new Array();
        for (i=1; i<20; i++)

        {

            public static var myTimer:Timer = new Timer(1500, 1
            myTimerArray[i] = myTimer;

        }

    }

}
```