1. Quick Start

To see how QFramework in action , go to http://www.100kas.com or http://www.teluminfo.hr , find live games and try them, all of the games can be downloaded from App Store, or Android Market.

Games can be played in your browser, mobile device or tablet. They are all developed using QFramework. This is document, as well as project will be upgraded, changed and enchanced.

2. Introduction

Qframework is designed to help develop 3d appliaction on mulitple platforms. Application can be simple game or user application which will use 3d for display its content. It is designed to be simple and fast to learn. It also has integrated server with clients designed for four platforms. In short, you can easily develop appliaction, deploy it on several major platforms. Languge for writing applications is javascript, while on server you can write application using javascript or python. Supported platforms are:

- IOS, (iPad, iPhone, iPod Touch)
- Android (version 2.1 and higher)
- Java Application with usage of JOGAmp
- Browser, Java Applet with usage of JOGAmp
- Browser, Html5 with WebGL support.

Inside source tree each platform has small Hello World example, and large tutorials program. Server example will be developed. To see framework online, go to project website. Easiest way to see kode and to test how to develop is to use Html5 client version. Here is suggestion how to do it:

- Fetch Google Chrome, and start it with *–allow-file-access-from-files*
- Open /Clients/Html5/tutorials/tutorials_test.html (or tutorials.html).
- Use your favorite text editor, to edit program files, located in /Clients/Html5/tutorials/res.

3. Getting q framework and setting up environment.

Framework is licensed under Apache license. It can be downloaded from:

- GitHub repository
- Complete archive.

3.1 Setting up environment for IOS

Requirements: MacBook and latest XCode.

Iphone code is located in /Clients/iPhoneQ. There are two clients that can be used as templates for project creation. Use HelloWorldQ or Tutorials project. Just load it in xcode, and start (if needed select ios platform ipad, iphone, or device). Project is created as universal application. Framework code is located in /Clients/iPhonQ/QFramework, so you need this folder also. This folder contains QFramework objective C and javascript source code.

3.2 Setting up Android environment

Requirements: Eclipse with latest Android sdk.

If you prepared eclipse for android development import /Clients/AndroidQ/QAnd, and /Clients/AndroidQ/tutorials , or /Clients/AndroidQ/HelloWorldQ into you workspace. You can import it from local directory, or repository. Start Tutorials, or HelloWorldQ as normal Android application. Each application is depended on QAnd project, in which android java code is located. Javascript framework code is located in assets/qres folder. Application code is located in /assets/res.

3.3 Setting up Java environment

Requirements: Eclipse with jogamp setup.

Please refer to http://www.jogamp.org how to setup jogamp. Jogamp is used for opengl support. Import /Clients/JavaQGl/Q and /Clients/JavaQGl/tutorials into your project workspace. Projects need jogamp as dependecies.

3.4 Setting up Html5 environment

Requrements: webgl enabled browser, Chrome is recomended.

Use /Clients/Html5Q/tutotorials/tutorials_test.html to test execution. Note that most of the browsers will not start javascript from local file system unless you specificly requested. For Chrome , start it with *–allow-file-access-from-files* , or upload files to server. For development you also need – decent text editor. Use browsers developer console and editor to debug and edit applications.

3.5 Setting up Server environment

Requirements: Windows or linux machine.

Server works on windows and linux machines (MacOs build is to be done). Windows precompiled binaries are located in /Bin/Server . Linux make file is provided, which is manually created. For server build you need:

- VisualStudio 2005
- Python 2.7
- Latest Boost library
- Latest V8 google javascript library.

4. Quick start

4.1 Introduction

All platforms have same amount of features, so if you create program on one platform, it will work automatically on other platforms. Program scripts are the same!

Program scripts are created in Java Script (ECMA), with addition of QFramework commands.

How framework works. When you issue q command, it will be put into queue, and executed in async way. This means, you will never get function value return (except for few util functions), and command interpreter will go on to next function. Here is small example:

Q.startUpdate(); // will start command session

Q.layout.areaSetText(“areaname1”, “Some Text”); // will set new text to text area

Q.layout.areaSetText(“areaname1”, “Other Text”); // will set new text to other text area

Q.sendUpdate(); //will send commands in queue.

As you can see from example, all commands are queued, and send after specific command. Queue session is started with Q.startUpdate(), and finished with Q.sendUpdate(). It is possible to have multiple sessions, as kind of session stack, but each session start has to finish with session end. For simple updates, most of commands have form that will automatically send command for update. So:

Q.layout.areaSetText_("areaname1", "Some Text"); // set new text to text area - NOW

Q.layout.areaSetText_("areaname1", "Other Text"); // set new text to other text area - NOW

This command form is introduced to simplify development.

Again to repeat, all Q commands are asynchronous. Its content is queued, and pushed to execution. It may sound strange, but this approach has advantages.

4.2 Script components

Very important – command interface documentation will describe components in brief way. Until documentation is not fully finished it is best to look for feature example in tutorials project, and to see how components are used. Use this only for quick reference.

Q Framework script interface is divided into several components. Main script language is javascript, while on server side python is also supported (full support will be finished soon). Each of the components together with examples will be described in next chapter. Following components are available:

- Q. – root component, with different commands.
- Q.layout – used to display layout components such as texts, grids, special areas. Layout is divided into separate areas.
- Q.textures – texture facility, for loading and naming textures.
- Q.objects – used to create , display and manipulate items in world.
- Q.models – used to define objects tempaltes.
- Q.sounds – for sound effect loading and playing.
- Q.camera – for modifying end setting current cameras properties.
- Q.settings – for reading and writing small amount of settings data.
- Q.env – modifying framework system properties.
- Q.handlers – for defining handler functions.
- Q.connect – for server communication
- Q.animations – for defining animation types
- Q.anim – for animation of objects
- Q.util – for small util functions
- Q.scores – component for leaderboards and achievements.

4.3 Q commands

All of the Q commands are access using Q global object. They are called as normal script commands, and internally transferred into asynchronous calls. Following commands are defined:

4.3.1 Q.startUpdate , Q.sendUpdate

```
Q.startUpdate();

Q.sendUpdate();
```

These functions are core of QFramework logice. *Q.startUpdate()* will start command session while *Q.sendUpdate()* will finish session, end send commands to execution. Every startUpdate command must match sendUpdate command. Best way is to start block of commands with startUpdate and finish it with sendUpdate. It is possible to start new command session before old one is closed. Most of the commands have two types of notation, with '_' and without it. If command has trailing '_' it means that it will be send to execution now. Framework will to start/send session automatically. These kind of functions are best used when performing single command line. Otherwise it is recommended to group commands in blocks, and bounded with start and send update.

4.3.2 Q.exec

```
Q.exec( <delay>, <script> );
```

Executes given script expression after delay. Delay is value in milliseconds. Script is string with javascript expression to be executed.

4.3.3 Q.event

```
Q.event( <id> , <delay>, <data> );
```

Sends async event to registered event handler, after delay in milliseconds. Event handler must be registered using *Q.handles* interface. Event handler will receive id and data of event. See *Q.handlers* for example.

4.3.4 Q.include

```
Q.include( <script_url> );
```

Loads script from given url. Currently only local relative url is supported. Command *Q.include('files/script.js');* will load script from folder relative of start script (main.js). Including of script can happened only once. If include command is called twice on same file, only first include will be taken into count. Use Q.load to parse and execute same script multiple times.

4.3.5 Q.load

```
Q.include( <script_url> );
```

Loads script from given url. Currently only local relative url is supported. Command Q.include('files/script.js'); will load script from folder relative of start script (main.js). This command will always load and execute script in current javascript scope.

4.3.6 Q.goUrl

```
Q.goUrl( <type> , < url> );
```

This function will request from local system to load external url. This function can be used to load url using local platform default handler. Currently only supported type is 'uri' and url is string that will be passed to system. For example command:

```
Q.goUrl_("url","http://78.47.124.235/promo/promo.html");
```

Will open given http url in browser. On android platform this example will open android market link:

```
Q.goUrl_("uri", "market://details?id=com.gameon.theprimes2");
```

4.3.7 Q.appendEvent

```
Q.appendEvent (<id>, <area>, <data>, <data2>, <data3>, <data4>);
```

Function will add event to session command queue. Internally all the functions are using this command to add data to queue. Each command has its id and values. Id is integer value while other values are string values. This function can be used to define new events and engine features. Otherwise it is best to use already predefined functions.

4.4 Q.layout

Layout component is used to display various predefined structures on screen. Layout that is displayed on screen is divided into pages with layout areas. Each page has its name and child areas. Layout area is defined with object called *LayoutArea.* There are several types of areas that will be described. Each type uses LayoutArea properties on different way, and represents data in space.

Object *LayoutArea* has following attributes:

- type – type of area can be one of following:
  - "text.label" – single line label text
  - "text.button" – single line text with icon
  - "text.mline" – multiline text
  - "text.mlinew" – multiline text with white space identation
  - "table.sgrid" – grid of elements.
  - "cards.deck" – stack of elements.
  - "cards.inhand" – simulates cards holded in hand
  - "cards.cards" – simulates cards on table
  - "layout.grid" – rectangle grid of elements.
  - "layout.back" – solid rectangle
- id – id of area, if not defined automatic value will be assigned.
- layout – layouting of area elements. Valid only on text.label values area:
  - hor – horizontal centering
  - none – default value.
- state – state of area. Valid values area
  - *visible – default value*
  - *hidden* – area is not shown and it is not active.
- bounds – width and height of area. Values are given as float values separated with comma, example "0.1,0.1".
- location – location of area, coordinates are location of area center and are given as x,y,z values, example "1.1,0.5,0.1". If not specified Z value is 0.
- rotation – rotation of area, values are rotation around x,y,z axis given as "x,y,z"
- scale – area scaling, given as scale values for x,y,z direction , "x,y,z"
- size – size of item data in area definition. Usage of this value varies of area type.

- display – defines display rule for this area. Can be:
  - "world", this is default value and camera used to see this element is from world camera data. This is rendered in the first pass, see camera component.
  - "hud" , renders area elements with hud camera definition. Best used to render overlay components, such as score texts, buttons etc.
- items – list of item definitions that will be placed in fields. Field are defined using field property or self defined in area creation.
- text – text value that will be displayed, used in text areas. Very important is to keep in mind that this field can displace underlaying JSON protocol. If such characters will occur use base 64 encoding. See dyntexts.js example as reference.
- onclick – defines script callback that will be used when user clicks on this area.
- onfocuslost – defines script callback that will be used when area loses user drag focus.
- onfocusgain – defines script callback that will be used when user enters this area while dragging or pressing this area.
- items – list of items that will be placed on field. Items are values separated with comma. See grid.js example.
- fields – field definition used in grids. This definition can define location of each field. See grid.js for example how to define.
- foreground – foreground color definition. Format is as following: "AARRGGBB,textid" where textid is optional parameter. Color is given in hexa values for alpha, red, green and blue. Texture is defined as texture name, or element in texture offset. This field may have different value for different area types. In text areas it is used for border color definition, while in sgrid it is used for texture of fields. See gris.js for example.
- background – background color definition. Format is as following: "AARRGGBB,textid" where textid is optional parameter. Color is given in hexa values for alpha, red, green and blue. Texture is defined as texture name, or element in texture offset. When this property is defined area has rectangle background rendered using color and texture information.
- border – defines border around area, used in text area. Valid value is:
  - "world"
- colors – color that are used to colorfy area. Colors values are used on corners on areas. It is four hexa color values separated with colon. See matrixeff.js for example.

This is only brief list of properties. Currently best way to use them is to go to tutorials project and to see feature that is needed and then see usage from code. To add area object to page and then to add page to layout following has to be done:

- Create array that will hold area objects.
- Create LayoutArea object and fill its properties.

- Add object to array.
- Call Q.layout.add to add areas to page. If page doesn't exists new page will be created.
- Call Q.layout.show to show page.

To see how to create pages see pages.js example.

Very important each area must have its unique id. When adding new area with id that already exists, old area is overwritten. It is best to use area ids, only when area has to be later referenced with this id (its properties changed).

4.4.1 LayoutArea object

When defined with id each area object can be access and some of its properties changed. Changing area properties will change its appearance. Areas are part of pages. By default when page is created and added to layout it is not visible.

4.4.2 Q.layout.add

```
Q.layout.add( <pageid>, <areas> );
```

Adds new page or updates existing one. Pageid is name of page, while areas is array of new area definitions. See examples for more info.

4.4.3 Q.layout.show

```
Q.layout.show( <pageid>);
```

Shows page on screen. This transition is immediate.

4.4.4 Q.layout.hide

```
Q.layout.hide( <pageid> );
```

Hides page from screen. Page can be shown again with Q.show command.

4.4.5 Q.layout.showAnim

```
Q.layout.showAnim( <pageid>, <type> );
```

This will show page with given id, while using transition type. Type is determined with animation type and delay in miliseconds. It is given as comma separated value. See pages.js for more info and available animation types. Transition will last for a defined delay in miliseconds.

4.4.6 Q.layout.hideAnim

```
Q.layout.hideAnim( <pageid>, <type> );
```

Hide page with animation effect. See showAnim.

4.4.7 Q.layout.popup

```
Q.layout.popup( <pageid> );
```

Shows page and locks focus on this page only. When page is hidden focus will be released. Best used for popup questions.

4.4.8 Q.layout.popupAnim

```
Q.layout.popupAnim( < pageid>, <type> );
```

Popups page and locks focus with animation.

4.4.9 Q.layout.clear

```
Q.layout.clear( <pageid> );
```

Clears page from layout and deletes all of its areas.

4.4.10 Q.layout.areaSetText

```
Q.layout.areaSetText( <areaid>, <text> );
```

Sets new text for area given with its id. Used on text areas to update its values. Text must be in format that will not mess JSON protocol. If not sure, use base64 encoding from Q.util.tob64. See dyntext.js for example.

4.4.11 Q.layout.areaSetLocation

```
Q.layout.areaSetLocation( <areaid>, <location> );
```

This will update area location. Complete area will be moved to new location. Location is given as "x,y,z" where x,y,z area float values of new coordinates.

4.4.12 Q.layout.areaSetBounds

```
Q.layout.areaSetBounds( < areaid >, <bounds> );
```

Sets new area bounds. Bounds are given as width, height and thickness value. Thickness is optional value. Values are float values given in string separated with comma.

4.4.13 Q.layout.areaSetRotation

```
Q.layout.areaSetRotation( < areaid >, <rotation> );
```

Sets rotation of area. Rotation is given as rotation around x,y, and z axis. Values are float values given in string separated with comma.

4.4.14 Q.layout.areaSetScale

```
Q.layout.areaSetScale( < areaid >, <scale> );
```

Scales area with values given in scale parameter. Scale parameter is string with three float values separated with comma.

4.4.15 Q.layout.areaSetItemScale

```
Q.layout.areaSetItemScale( < areaid >, <index> , <scale> );
```

This will scale single item in area. Item id is given as index.

4.4.16 Q.layout.areaSetItems

```
Q.layout.areaSetitems( < areaid >, <items> );
```

Sets new area items information. Item information is given as list of item values separated with comma. Item definition is as follow:

- `[i]itemset.id`
- `[t]textvalue`

It [t] is used text is rendered in field. If [i] is used system will search for item in defined test. See main.js how to define and sgrid.js how to set items.

4.4.17 Q.layout.areaSetItem

```
Q.layout.areaSetItems( < areaid >, <index>,  <item> );
```

Sets item in area on field with specified index. Item definition is same as for areaSetItems, but this is single value.

4.4.18 Q.layout.areaSetBackground

```
Q.layout.areaSetBackground( < areaid >, <background> );
```

Updates area background. When this function is applied area representation is changed and updated with new background. Area background is defines as two values separated with comma. Fist value is AARRGGBB color value, while second is texture identification. Texture can be name of texture, or name of texture together with its offset values. Offset is used to display only part of texture. See how exit button is defined in tutorials.

4.4.19 Q.layout.areaSetOnclick

```
Q.layout.areaSetOnClick( < areaid >, <script> );
```

Sets handler for click event. Click event is when user presses area (tap and release, or press and release). Handler can have three forms:

- *Single string* – which is send to data handler. To use it define string value and it will be send onData handler if defined.
- *Javascript function* – define with *js:functionName* , areaid and field index will be passed as arguments to this function.
- *Javascript expression* – define it with js:jsexpression; , don't forget to put ; at end of expression. Expression will be executed upon event.

To clear area handler for click, call this function with empty string.

4.4.20 Q.layout.areaSetOnFocusGain

```
Q.layout.areaSetOnFocusGain( < areaid >, <script> );
```

This will set event handler when user gives focus on this area. Focus is given when user enters area while dragging or starting to press button. See cards.js for more info. Format is same as for onclick handler.

4.4.21 Q.layout.areaSetOnFocusLost

```
Q.layout.areaSetOnFocusLost( < areaid >, <script> );
```

This will set event handler when user loses focus on this area. Focus is lost when user leaves area while dragging or releases press above area. See cards.js for more info. Format is same as for onclick handler.

4.4.22 Q.layout.areaSetState

```
Q.layout.areaSetState ( < areaid >, <state> );
```

Updates area state. Used to hide or show area. State can be *visible* or *hidden string.*

4.4.23 Q.layout.areaDelete

```
Q.layout.areaDelete( < areaid >);
```

Deletes area and removes it from its page.

4.4.24 Q.layout.editText

```
Q.layout.editText( < defvalue>, <callback> );
```

Starts external text editor. Defvalue is default editor value that will be edited. Callback is name of javascript function that will be called upon every character change of text field. Function has two arguments, where first is current text edit value, and second parameter is set to notify user if editing is finished. If this parameter has value 1 this is final call and editing is finished.

4.4.25 Q.layout.areaClearItems

```
Q.layout.areaClearItems( < areaid > );
```

Clears area item fields.

4.4.26 Q.layout.itemPlace

```
Q.layout.itemPlace( <itemtype>,<id>,<areaTo>, <indexTo> );
```

This function will place item on area. Item is determined with its type and id (cards.31). This function is used to update single item in area.

4.4.27 Q.layout.itemMove

```
Q.layout.itemMove( <areaFrom>, <indexFrom>, <areaTo>,<indexTo> );
```

This function will move index from one area to another area. Item moving will not be animated.

4.4.28 Q.layout.itemRemove

```
Q.layout.itemRemove( < areaid >, <index> );
```

Function will remove item from area. Item is placed on area field denoted with index value.

4.4.29 Q.layout.itemMoveA

```
Q.layout.itemMoveA( <areaFrom>, <indexFrom>, <areaTo>,<indexTo>,
<type>, <delay> );
```

Function will move item from one area to another area using given animation. Delay is time value in milliseconds and determines animation length. See cards.js for example and animation component about animation types.

4.4.30 Q.layout.itemsAnim

```
Q.layout.itemsAnim( < areaid >, <type>, <delay> );
```

Animates items on area with given animation type. Animation delay is time in milliseconds. If it is defined as “delay,repeat” repeat value is used to set number of animation repeats.

4.4.31 Q.layout.itemAnim

```
Q.layout.itemAnim( < areaid >, <index> , <type>, <delay> );
```

Animates single area item.

4.5 Q.textures

Textures facility is used to load textures from local file system, gives them properties and use in rendering. Recommended texture file format is png. Currently framework is not using any kind of texture compression, so user must take care of resource allocation.

Texture can be created using *Texture* object or Q.textures component.

Texture object has following properties:

- *name* – name of texture
- *file* – relative path to texture.

4.5.1 Q.textures.newFromTheFile

```
Q.textures.newFromTheFile( <textid> , <filepath> );
```

Function will create new texture from file located on filepath. Filepath requires file extension to be also part of path. It is recommended to use rectangular png files with width/height on potent ion of 2. Textid will be id of texture that will be used in other functions. Framework will automatically reload textures if surface is lost or recreated.

4.5.2 Q.textures.add

```
Q.textures.add( <textid> , <textarray> );
```

Adds new textures from array of Texture objects. Use this function to load list of textures on fast way. See main.js of examples to to use this function.

4.6 Q.sounds

Sounds component is used to play small sound effect. Sound effects should be located in local folder directory of scripts (relative or in same folder as main.js). Each platform has different type of sound effect that supports (Android mp3, IOS caf, Java wav). Because of that when naming sound effect extension should not be used. In future this will be fixed.

4.6.1 Q.sounds.newFromFile

```
Q.sounds.newFromFile( <soundid> , <filepath> );
```

Function will load new sound effect from file and associate given id. Any future operation on this sound will be done using its sound id. Filepath is path of file relative to main folder (location of main.js). For now don't use extension while loading sound.

4.6.2 Q.sounds.play

```
Q.sounds.play( <soundid>);
```

Plays sound effect.

4.6.3 Q.sounds.volume

```
Q.sounds.volume( <volumelevel>);
```

Sets volume level, where volume level is value from 0-100.

4.6.4 Q.sounds.mute

```
Q.sounds.mute( <muteonoff>);
```

Mutes sound playing. Set value to 1 to mute all sounds.

4.7 Q.models

Model is item template used to display specific shapes on screen. To use models , such as cards, figures they should be constructed first using this component. For now this component can create only few element types, which are already predefined. In future more shapes will be available and object import will be created.

3.7.1 Q.models.newFromTemplate

```
Q.models.newFromTemplate( <modelid> , <template>);
```

Creates new model based on its template. Template can be one of the following values:

- *sphere*- creates sphere, with radius of 0.5 (diameter 1.0)
- *cube* – creates simple cube with side of 1.0
- *cylinder* – creates cylinder model with diameter of 1.0 and height of 1.0
- *plane* – creates rectangular plane with side of 1.0
- *card52*- creates object specialized for card display. Has two sides. See cards.js for example.

All the objects have size of 1.0 and their shapes are centered.

3.7.2 Q.models.setTexture

```
Q.models.setTexture( <modelid> , <textureid>);
```

Maps texture to model. Texture should be already loaded with texture component.

3.7.3 Q.models.create

```
Q.models.create( <modelid>);
```

Creates geometry of model with modelid. This will actually create model geometry in system memory. This function should be called last in chain of model creation.

3.7.4 Q.models.setSubmodels

```
Q.models.setSubmodels( <name> , < value >);
```

This function is used to do additional object creation. Function will inform framework that this object may have additional objects inside single textures. If we create object with name *cards*, and set submodel value to 64, we can reference submodels using relative index of object, such as cards.10 or *[i]cards.10* if we are setting object as area item. See cards.js for more details. Using this approach we can use single texture to display more objects, and with this increase program performance. Value

can be single integer value which is power of 2. Additional value can be added (“val1,val2”) which is used in special card52 object. It is used to render background of card. See cards.js.

3.7.5 Q.models.add

```
Q.models.add( <arrayofmodels>);
```

This function is used to add array of models in single command. To create model object create *Model* object (*var model = new Model();*) and set its properties. Following properties are defined:

- *name* – name of model, its modelid.
- *template* – template used to create model
- *texture* – texture id used to map model.
- *submodels* – value how to slice texture into additional submodels.

4.8 Q.settings

Settings component is used to store simple data models into local device storage. Data can be integer, string or array represented as string. Single storage is available for application. It is only way how application can store persistent data.

3.8.1 Q.settings.open

```
Q.settings.open();
```

Opens settings file for reading and writing. Should be called before reading or writing data.

3.8.2 Q.settings.save

```
Q.settings.save();
```

Performs data saving with local storage. To save data, this function must be called after write commands are executed. This will actually store data to local storage. After this command, to write again, open command should be called again.

3.8.3 Q.settings.writeInt

```
Q.settings.writeInt( <key>, <intvalue>);
```

Saves integer value into named slot. Name of slot is given with key parameter.

3.8.4 Q.settings.writeString

```
Q.settings.writeString ( <key>, <stringvalue>);
```

Saves string value into named slot. Name of slot is given with key parameter. If string may cause break of JSON protocol use base 64 encoding to solve this issue.

3.8.5 Q.settings.loadInt

```
Q.settings.loadInt( <key> , <variable>);
```

Loads integer value from storage and stores its value into variable.

3.8.6 Q.settings.loadString

```
Q.settings.loadString( <key> , <variable>);
```

Loads string value from storage and stores its value into variable.

3.8.7 Q.settings.loadArray

```
Q.settings.loadArray( <key> , <variable>);
```

Loads string value from storage and stores its value into array variable. This function can be executed only if array was stored using its *toString* command.

4.9 Q.camera

This components sets location of camera for rendering elements. Framework has two stages of rendering. In first stage all elements that are placed in world will be rendered. In second stage, elements that are placed in hud domain (see display property of areas) will be rendered. Hud elements will be always drawn on top of world elements. Each domain has its own camera defined. Best practice is to render world objects using camera placed somewhere in the world, while hud elements are rendered with camera placed orthogonal on xy plane. In this way hud elements can be control button, texts, while camera for the world elements can be changed. To make development of applications easier for different screen sizes following properties are defined:

- Q.layout.canvasw - width of screen
- Q.layout.canvash -height of screen
- Q.layout.worldxmin –minimum x coordinate based on current world camera
- Q.layout.worldxmax –maximum x coordinate based on current world camera
- Q.layout.worldymin –minimum y coordinate based on current world camera
- Q.layout.worldymax –maximum y coordinate based on current world camera
- Q.layout.hudxmin –minimum x coordinate based on current hud camera
- Q.layout.hudxmax –maximum x coordinate based on current hud camera
- Q.layout.hudymin –minimum y coordinate based on current hud camera
- Q.layout.hudymax –maximum y coordinate based on current hud camera

Each if this property is automatically changed if camera parameter is changed.

3.9.1 Q.camera.fit

```
Q.camera.fit( <width>, <height>);
```

Automatically positions camera on coordinate orthogonal to xy plane, in a way that minimum x coordinate is –width/2, maximum x coordinate is width/2, y minium coordinate is -height/2, and maximum y coordinate is height/2.

3.9.2 Q.camera.set

```
Q.camera.set( <xeye>,<yeye>,<zeye>, <xat>,<yat>,<zat>);
```

Positions camera on coordinate determined with eye coordinates. Camera direction is pointed at at coordinates.

3.9.3 Q.camera.fitHud

```
Q.camera.fitHud( <width>, <height>);
```

Automatically positions hud camera on coordinate orthogonal to xy plane, in a way that minimum x coordinate is –width/2, maximum x coordinate is width/2, y minium coordinate is -height/2, and maximum y coordinate is height/2. Use Q.layout.hud* coordinates to place objects relative to screen bounds, see examples.

3.9.4 Q.camera.setHud

```
Q.camera.setHud( <xeye>,<yeye>,<zeye>, <xat>,<yat>,<zat>);
```

Positions hud camera on coordinate determined with eye coordinates. Camera direction is pointed at *at* set of coordinates.

4.10 Q.env

There are several framework system variables that can be set and used to perform certain actions. They are accessed using env component.

3.10.1 Q.env.set

```
Q.env.set ( <name>,<value>);
```

Sets system environment property on value. Following value elements are available:

- *textparam* - sets current font texture parameters. Value is represented with five values separated with comma (string). Values are *u1,v1,u2,v2,cp.* UV values represents texture mapping on each letter. If values are larger texture coordinate for each letter will be moved to center. Values should be between 0 and 0.5. CP determines how much will each letter overlap next one.
- *touch* – Enables or disables touch events on whole screen. Value can be *on* to enable touch events and *off* to disable touch events.

4.11 Q.connect

Clients can have single connection to server that supports Q protocol. Before receiving data from server, client must join one of the rooms. After that client will automatically receive and process server commands. While server can do everything with client layout, client can only send data to server.

3.11.1 Q.connect.connect

```
Q.connect.connect( <address>, <callback>);
```

Connects client to server on address (ipaddress:ipport). Callback is javascript function which will be called upon finishing request. Callback has one parameter which is set to 1 if connection is success.

3.11.2 Q.connect.disconnect

```
Q.connect.disconnect();
```

Disconnects client from current server connection.

3.11.3 Q.connect.join

```
Q.connect.join(<roomid> , <username> , <callback> );
```

Joins application in one of the server rooms.

3.11.4 Q.connect.send

```
Q.connect.send(<data> );
```

Sends data to room on which user is joined.

More details will be available when server examples are available. For now see Sedmice project example.

4.12 Q.handlers

Each script can register event handlers. Event handler will occur when specific command or situation is reched. Some events will happened only on server scripts. To register handler simply push new callback to handler array. For example:

```
function game_onGameEvent(type, userdata)

{
}

Q.handlers.onEvent.push(game_onGameEvent);
```

Will define handler function for event. When Q.event(10,1000, ”somedata”); command is issued after 1 second your handler game_onGameEvent will be called with type value of 10, and userdata “somedata”.

3.12.1 Q.handlers.onInit

Registered function that will be called just before script is started. Can be only used on server.

3.12.2 Q.handlers.onInfo

This handler will be called periodically to send server info about script status. Can be used on server only. User should return string as result.

3.12.4 Q.handlers.onEnd

Not used.

3.12.5 Q.handlers.onEvent

Function that is registered as event handler will be called as answer to Q.event command. Can be used on server and client.

3.12.6 Q.handlers.onUserJoined

Used to notify script that new user joined. It is used only on server.

3.12.7 Q.handlers.onUserLeft

Used to notify script that user left the room. It is used only on server.

3.12.8 Q.handlers.onData

It is called when user sends data. It is used only on server.

4.13 Q.util

This component implements several util function that may help development. Unlike other functions they are simple javascript utility functions.

3.13.1 Q.util.touchonoff

```
Q.util.touchonoff( <delay>);
```

Turns touch events off (screen touching) for delay defined in milliseconds. Uses Q.env.touch function.

3.13.2 Q.util.indexOf

```
Q.util.indexOf( <array> , <element>);
```

Returns index of element in array. Returns -1 if no element is found.

3.13.3 Q.util.arrayRemove

```
Q.util.arrayRemove( <array> , <element>);
```

Removes element from array.

3.13.4 Q.util.tob64

```
Q.util.tob64( <string>);
```

Transforms string to base 64 and adds prefix #64# in front of string to be recognized by framework.

4.14 Q.objects

This facility creates objects that can be placed in world space. Object will be created from model type. Model type has to be defined using model facility or predefined in framework. For now only basic object types are defined (cube, sphere, plane). See objects.js for example.

3.14.1 Q.objects.create

```
Q.objects.create( <newobjectid>, <modelname>);
```

Creates new object based on its model name. All other calls to this object will be with referencing it with object id. Default size of object is 1.0, 1.0, 1.0. Default place is on zero coordinate, and its state is hidden.

3.14.2 Q.objects.place

Q.objects.place(<objectid>, <location>);

Places object in space to new location. Location is defined as x,y,z coordinates. Coordinates are float numbers separated with comma, and given as string parameter.

3.14.3 Q.objects.remove

Q.objects.remove(<objectid>);

Removes object from world space. This will also destroy object.

3.14.4 Q.objects.scale

Q.objects.scale(<objectid>, <scale>);

Scales object to new size. Scale is defined as x,y,z scale values. Sizes are float numbers separated with comma, and given as string parameter.

3.14.5 Q.objects.state

Q.objects.state(<objectid>, <state>);

Sets new object state. State values can be :

- visible – to show object
- hidden – to hide object

3.14.6 Q.objects.texture

Q.objects.texture(<objectid>, <textureid>);

Sets new object texture. Texture must be loaded using Textures component.

3.14.7 Q.objects.add

Q.objects.add(<arraofobjects>);

Adds array ofobject. Create series of WorldObject objects, add them to array and pass into function as parameter. WorldObject has following properties:

- name – name of object that will be created
- template – model template for object
- location – new location of object
- bounds – object bounds
- texture – object texture
- state – state of object, visible or hidden

See objects.js for more info. This component will be further improved, by adding object animations, batch rendering.

4.15 Q.anim

This component is used to animate existing object. See objects.js for example. This component will be more developed.

3.15.1 Q.anim.move

```
Q.anim.move( <objectid>, <location>, <delay>, <callback>);
```

Moves object to new location. Location is defined as x,y,z coordinates. Delay is movement delay is miliseconds. If format *"delay,repeat"* is used, repeat will determing how many times animation is repeated. If repeat is <0 it will notify framework that second part of animation is going back to start coordinate.

3.15.1 Q.anim.rotate

```
Q.anim.rotate( <objectid>, <rotation>, <delay>, <callback>);
```

Rotates object around itself.

This component will be more developed.

4.16 Q.animations

Adds new animation template.

4.16.1 Q.animations.add