Tallinna Ülikool
Informaatika Instituut

Animeerimine JavaScriptiga

Seminaritöö

Autor: Heindrig Paabut
Juhendaja: Andrus Rinde

Tallinn 2013

Sisukord

Sissejuhatus

Esmased animatsioonid veebilehtedel olid animeeritud GIFid (*Graphics Interchange Format*), pildid, mis olid piisavalt väikesemahulised, et neid suudeti veebilehele kuvada ka aeglaste modemite ning kehva internetiühendusega. Animeeritud GIFidega loodi lehtedele bännerid, üksikuid liikuvaid elemente ning hiljem ka avatare. (Buck, 2012) Järgnev samm veebianimeerimises oli *FutureSplash Animator*i kasutamine, mis on eelkäijaks *Flash'*ile. 1996. Aastal, müüdi *FutureSplash Macromeda'*le ning sellest sai *Flash*, mille kasutus animatsioonide ning mängude loomises laienes kiiresti. 1999. Aastal lisati *Flash'*i neljandale väljalaskele *ActionScript*, mis võimaldas luua keerukamaid ja interaktiivsemaid animatsioone. Väidetavalt tänu *ActionScript'ile* muutus *Flash* veelgi populaarsemaks ning muutus masside poolt kasutatavaks. (Rocheleau, 2012)

Flash on aga oma edu kaotamas, sest on kinnine platvorm ning nõuab rohkelt jõudlust. Neil põhjustel otsitakse alternatiivi *Flash*'ile, milleks on saamas *JavaScript*, mis pakub nüüd ka üha enam ja enam animeerimise võimalust. Näiteks taimeritele loodud alternatiivi ehk *requestAnimationFrame* API üha laienev kasutus, mis võimaldab mugavamat animeerimist. (Zim, 2012) Teiseks suureks muutuseks on HTML5-ga kaasnev algselt *Apple*'i poolt loodud ning tutvustatud uus element *canvas* ehk lõuend. Lõuend võimaldab *JavaScript'*iga joonistamist ning animeerimist. (Marinacci) Kolmas põhjus, mis suurendab *JavaScript'*i tähtsust animeerimises on samuti lõuendile toetuv, rohke raamistike ja teekide hulk. Need on spetsiaalselt välja töötatud animeerimiseks lõuendil, sarnaselt *ActionScript'*ile, mis manipuleerib *Flash*'i lava. (McBride, 2011)

Käesolev seminaritöö käsitlebki üht raamistikku ja kahte teeki, mis on autori poolt välja valitud väljamõeldud kriteeriumide ning Autori subjektiivse eelistuse põhjal.

Teema valis autor lähtudes oma huvist animeerimise vastu ning ka erialasest vajadusest õppida *JavaScript'*i. Lisaks üha suurenev *JavaScript'*i kasutus animeerimises, mängu loomes ja üldiselt veebiarenduses ehk alternatiivi leidmine *Flash'*ile ning *ActionScript'*ile.

Antud töös tehakse ülevaade puhtast *JavaScript'*ist ning *JavaScript'*i põhistest animeerimiseks mõeldud raamistikust ja teekidest. Autor loob mõned tüüpnäited, et anda selgemat ettekujutust *JavaScript'*i potentsiaalist animeerimisel ning raamistike ja teekide tugevustest ning nõrkustest.

Töö eesmärk on anda algajale *JavaScript'*i ning vilunud *ActionScript'*i animeerijale ülevaade *JavaScript'*iga animeerimise võimalustest, tutvustada mõnda lõuendipõhist raamistikku ja teeki. Eesmärgi täitmiseks loob autor väljavalitud raamistiku ning kahe teegiga kaks näidislahendust, mis illustreerivad väljamõeldud kriteeriumeid. Nimelt katsetab autor loodavate programmijuppide näitel tüüpanimatsioonide töövõimet antud seminaritöös käsitletavate raamistiku, teekide ja puhta *JavaScript'*i abil.

1 Ülevaade JavaScriptist

JavaScript, algse nimega *Mocha*, loodi aastal 1995. Sellest ajast saadik on *JavaScript* olnud kiires arengus ning kehtestatud *ECMAScript* (*European Computer Manufacturers Association*) standardiks. (Young, 2010) *JavaScript*'i kasutati algselt veebilehe interaktiivsemaks muutmiseks, et kasutaja ei peaks peale igat päringut lehte uuendama, vaid et uuendus toimuks kohe. (Chapman) Näiteks, kui täideti registreerimise blanketti, siis tulid vead välja viivitamatult (parool ei kattunud või kasutajanimi on juba olemas), seega ei vajanud veebileht värskendamist ja see muutis kogu kasutajakogemuse mugavamaks.

JavaScript'i massidesse süvenemine sai alguse 2000. keskpaiku, kui hakati *JavaScript'*i avatud lähtekoodi (*open source*) laialdasemalt kasutama ning seda sõltumatult suurfirmadest arendama. Tekkisid esimesed *JavaScript*'i põhised raamistikud ning teegid, mis lisasid uusi võimalusi ning olid mõeldud mingi kitsama valdkonna lihtsamaks haldamiseks, näiteks animeerimise raamistikud. Ühtlasi aitasid need muutused *JavaScript*'il saada üha laiemalt kasutatavaks programmeerimisekeeleks.

HTML5 tulekuga on *JavaScipt* muutunud veel olulisemaks vahendiks veebiarenduses ning animeerimises. Dokumendi objektimudeli (*Document Object Model* - DOM) manipuleerimine *JavaScript'*i või mõne *JavaScript'*il põhineva raamistiku või teegi abil on parimaid alternatiive *Flash'*i lavale ning *ActionScript'*ile.

*JavaScript'*i suurim tugevus on sidumisvõimalus erinevate programmeerimiskeeltega ning rohke raamistike ning teekide hulk, mis võimaldavad või lihtsustavad teatud ülesannete täitmist. Animeerides on *JavaScript'*i eelisteks väike koodide hulk, loodud animatsioonidel on väiksem maht võrreldes *Flash'*iga ning võimalus animeerida ka lõuendiväliselt.

Suurimaks nõrkuseks on see, et *Javascript* on algselt keerukam kui *Flash*. Lihtsamaid animatsioone luues saab *Flash'*is läbi ka *ActionScript'*ita ehk lihtsalt lavale joonistades ning kaader-kaader haaval liigutades, kuid *JavaScript'*is on isegi lihtsama animatsiooni loomiseks vaja koodi kirjutada.

1.1 JavaScriptiga animeerimine

Animeerimiseks loetakse põhiliselt objektide liigutamist (*move*), kallutamist (*skew*) ja pööramist (*rotation*). Nende kolme animeerimisvõtte kombineerimisel hiire või nupu tuvastusega ning objektide läbipaistvuse muutmisega on võimalik luua põhimõtteliselt kõiki

animatsioone, alustades objektide lihtsate liikumistega punktist A punkti B ning lõpetades veebipõhiste mängudega.

Põhiliselt luuakse animatsioone HTML5 lõuendil, kuid on ka võimalik animeerida lihtsalt dokumendis, nt menüüsid, avanevaid vorme, liikuvaid elemente kogu taustal.

JavaScript'iga animeerimiseks on kaks põhilist moodust: esiteks *requestAnimationFrame* API ning teiseks taimerite kasutamine. Esimene neist kahest on uus meetod, mis on välja töötatud *Mozilla* poolt ning võimaldab suuremat töökindlust, kuna ei toetu millisekundite lugemisele, mida on võimalik häirida, näiteks kui arvuti jõudlus on mitmete avatud ning töötavate programmide poolt kasutatud. (Irish, 2011)

Mõlemal juhul saab saavutada sarnase tulemuse, ehk siis liikuvad objektid lõuendil.

2 Animeerimist lihtsustavad teegid ja raamistikud

Framework - raamistik on objektorienteeritud süsteemide puhul objektiklasside hulk, mis annab kasutajale või programmile omavahel seotud funktsioonide kollektsioon.

Library – teek on valmiskompileeritud alamprogramm, mida põhiprogramm saab kasutada. Need alamprogrammid, mida kutsutakse ka mooduliteks, salvestatakse objektikoodidena.

Võrreldes puhtakujulise *JavaScript*'i kasutamisega on raamistikke ning teeke lihtsam kasutada. Suuresti on põhjuseks see, et raamistikud ja teegid on loodud mingi ülesande spetsiifilisemaks lahendamiseks. Sellest tulenevalt on raamistikel ja teekidel ka konkreetsem dokumentatsioon ning eesmärgist lähtuv näidete kogu. Antud kontekstis on raamistike ja teekide dokumentatsioonis ja õpetustes rõhutud animatsioonidele, kujundite loomisele, sündmuste kuulamisele ja muule relevantsele animeerimises.
Käesolevas seminaritöös käsitlemiseks otsustas autor valida teegid ja raamistiku järgmiste kriteeriumite alusel.

- Lõuendi manipuleerimine (HTML5 on tuleviku standard, mille üks elemente on lõuend ning tänu sellele muutub lõuend ka erinevate veebisirvijate poolt üha enam toetatavaks.)
- Pildi animeerimine (Igal raamistikul ja teegil peab olema võimalus importida pilt lõuendile ning hiljem seda objekti animeerida.)
- 2D animatsioonid (Peab pakkuma mugavaid vahendeid 2D animatsiooni loomiseks. Käesolevas seminaritöös piirdub autor tasapinnaliste animatsioonidega, seetõttu otsustas välja jätta 3D animatsiooni toetavad raamistikud ja teegid.)
- Lihtne süntaks (Autor tunnistab, et tegemist on subjektiivse kriteeriumiga ning leiab oma kogemustest lähtudes, et on lihtne ning kujutab ette, kuidas algajad võiksid sellest aru saada.)

Vastavalt neile autori poolt määratud kriteeriumitele, tänu oma võimekusele lõuendi manipuleerimises ja õpetuste rohkusele ning dokumentatsiooni arusaadavuse tõttu osutusid valituks raamistik *KineticJS* ja teegid *FabricJS* ning *SVGjs*. Ühtlasi on valituks osutunud

raamistik ja teegid autori loodud testülesannete põhjal kõige põhjalikumad ja lihtsamini kasutatavad.

2.1 KineticJS

KineticJS on raamistik, mis on loodud lõuendielemendi töötlemiseks. Raamistik on järjepidevas arengus ning kasutajaid hoitakse uute versioonide tuleku, kaasnevate uuenduste ning tulevikus plaanitavate uuendustega kursis *github*i vahendusel (Drowell)

KineticJS võimaldab lõuendile joonistada kokku üksteist erinevat kujundit, milleks on ristkülik (*rectangle*), ring (*circle*), sektor (*wedge*), joon (*line*), hulknurk (*polygon*), pilt (*image*), sprite (*sprite*), tekst (*text*), laik (*blob*), vabakujund (*custom*) ning animeerida neid kujundeid peaaegu, et igal võimalikul moel. Lisaks on võimalik salvestada lõuendi sisu pildina oma valitud vormingus. On võimalik luua ka kohandatud ehitus allalaetavale lähtekoodile (*source code*), kasutades näiteks ainult konkreetses projektis vajaminevaid osi.

Raamistik on väga lihtsa süntaksiga, kergesti loetav ning arusaadav, sellest tulenevalt ei vaja *KineticJS*'i kasutamine väga palju varasemaid kogemusi *JavaScript*'iga. Autori arvates on antud raamistik algajale kasutajale üks parimaid variante tänu oma lihtsusele ning väga heale ning põhjalikule õpetustelehele, mis toob välja valdava osa *KineticJS* võimalustest koos illustreeritud näidetega.

2.2 FabricJS

FabricJS on lõuendielemendi manipuleerimiseks loodud teek, mis võimaldab kasutajal lõuendile joonistada ning oma loomingut liigutades, keerates või skaleerides animeerida. *FabricJS*'i üks suuremaid plusse on see, et sarnaselt *Flash*'ile käsitleb see lõuendil asuvaid joonistusi/pilte objektidena. Ühtlasi on võimalik lõuendi sisu salvestada sobivas vormingus pildina ning lisaks ka *JavaScript*'i objekti märkmete (*JSON – JavaScript Object Notation*) või skaleeritava vektorgraafikana (SVG – Scalable Vector Graphics). Teegi lähtekoodi alla laadides, saab valida kasutamiseks vaid olulised osad lähtekoodist ja sellega vähendada allalaaditava faili mahtu, samuti nagu *KineticJS*'i puhul.

Teek ise on lihtsa ning kergesti loetava süntaksiga, mis sobib hästi algajatele. *FabricJS*'i õppimiseks on teegi kodulehel õpetuste lehel rohkelt demosid, kergesti hoomatav dokumentatsioon ning algajatele neljaosaline detailne seletav õpetus, mis katab esmased ning põhilised *FabricJS*'i võimalused ja rakendusalad.

2.3 SVGjs

SVGjs on väikesemahuline teek, mille ülesandeks on manipuleerida lõuendit ning lõuendil olevaid objekte. *SVGjs'*i on autori arvates otstarbekam kasutada lihtsamate ja pisemate (lihtsa graafilise objekti liikumine punktist A punkti B) animatsioonide loomiseks. Teek saab hakkama ka keerukamate animatsioonide loomisega, kuid nõuab rohkem süvenemist ning erinevate pistikprogrammide kasutamist. Antud teegi süntaks võimaldab võrreldes eelpool mainitutega oluliselt lühemat koodi. Oma mahust hoolimata võimaldab *SVGjs* põhilisi manipulatsioone ehk liigutamist, skaleerimist ja keeramist.

Dokumentatsioon on nagu eelnevatelgi tööriistadel põhjalik ning kergesti loetav. Lisaks on ametlikul kodulehel testleht, kus on erinevate objektide peal proovitud põhilised *SVGjs* poolt pakutavad võimalused. Kuna teek on võrdlemisi uus ning alles beeta arendusjärgus võib tulla rakendusliideses ette muudatusi.

3 Animatsioonide näited

Järgnevalt demonstreerib autor, kuidas lahendada tüüpilisemaid animeerimisülesandeid vaadeldud raamistiku ja teekide abil. Võrdlusena lahendatakse sama ülesanne võimalusel ka puhta javascriptiga vahenditega.

Näidete valiku suurimaks mõjutajaks olid väljamõeldud kriteeriumid, tänu millele valiti välja ka sobilikud teegid ja raamistik.

Kõige tüüpilisemad animatsioonid on seotud objektide lihtsa liikumisega, mis sageli käivitatakse hiiresündmustele reageerides.

3.1 Hiire tuvastus ning sündmused

Esimeses autori poolt loodud näites keskendutakse hiire liikumise ning nupuvajutuse tuvastamisele. Ülesandes luuakse neli ruutu, igale ruudule on lisatud erinev efekt (läbipaistvuse muutumine, kujundi hävitamine, kujundi suuruse muutmine), mis käivitatakse vastava hiireliikumise või nupuvajutuse peale (klikk, topelt-klikk, hiir kujundi peal, hiir kujundilt maha). Loodavatele ruutudele lisatakse atribuudid (suurus, positsioon, värv, konkreetse ruuduga toimuv muudatus), seejärel luuakse funktsioonid, mis viivad läbi atribuutide muudatused.

Autor valis üheks testülesandeks just sellise programmi, kuna animeerimises ning veebiprogrammeerimises kasutatakse väga sageli hiiretuvastust, et muuta veebileht või animatsioon interaktiivsemaks ning muudetakse veebilehe või animatsiooni kasutamine vahetumaks ning lihtsamaks.

Ekraanitõmmis 2. Algne seisund.

Ekraanitõmmis 1. Sesinud hiire sündmuste järgselt.

3.1.1 JavaScript

Ülesande täitmiseks defineeritakse ruut efektitüübi ning muutusega, mis rakendub sündmuse järgselt.

Koodinäide 1. Ruudu defineerimine.

```
var myRect1 = new Rectangle({
  marginTop: 0,
  resizeTimes: 2,
  FXType: 'decrease'
});
```

Hiire liikumise või nupuvajutuse järel kutsutakse välja funktsioon, mis viib läbi vastava muutuse. Antud funktsioon rakendub konkreetsele ruudule, kui on tuvastatud selle asukoha.

Koodinäide 2. Funktsioon, millega muudetakse ruutu.

```
function triggerFX(rectangle) {
  if (rectangle.width != rectangle.originalWidth && rectangle.height != rectangle.originalHeight) {
    rectangle.width = rectangle.originalWidth;
    rectangle.height = rectangle.originalHeight;
    return rectangle;
  } else {
    switch (rectangle.FXType) {
      case 'decrease':
        rectangle.width = rectangle.width / rectangle.resizeTimes;
        rectangle.height = rectangle.height / rectangle.resizeTimes;
        return rectangle;
    }
  }
}
```

Autori kogemusest lähtudes oli keerukaim versioon antud ülesannet lahendada kasutades puhtakujulist *JavaScript*'i. Kuna koodi tuleb rohkem ning funktsionaalsus vajab täpsemat väljakirjutamist, näiteks kujundi kuvamisel, tuleb iga muutus arvesse võtta ning see vastavalt lõuendile paigutada.

Koodinäide 3. Funktsioon, millega ruut lõuendile joonistatakse.

```
function render(rectangle) {
  canvasContext.globalAlpha = rectangle.opacity;
  canvasContext.fillStyle = rectangle.color;
  canvasContext.fillRect(rectangle.marginLeft, rectangle.marginTop, rectangle.width, rectangle.height);
  canvasContext.stroke();
}
```

Üks esimesi probleeme oli ristküliku asukoha tuvastamine, mis osutus kordades keerukamaks, kui võrrelda teekide või raamistikuga.

Koodinäide 4. Funktsioon, millega tuvastatakse ruudu asukoht.

```
function correctPosition(rectangle, event) {
  var mouse = getMousePosition(event);
  if (
     mouse.x > parseInt(rectangle.marginLeft) &&
     mouse.x < (rectangle.marginLeft + rectangle.width) &&
     mouse.y > parseInt(rectangle.marginTop) &&
     mouse.y < (rectangle.marginTop + rectangle.height)
    ) {
     return true;
  } else {
    return false;
  }
}
```

Lisaks korduv lõuendi puhastamine, otseselt pole see probleem, kuid on tüütu lisafunktsionaalsus.

Koodinäide 5. Funktsioon, millega puhastatakse lõuend.

```
function clear(rectangle) {
  canvasContext.clearRect(
    rectangle.marginLeft,
    rectangle.marginTop,
    rectangle.width,
    rectangle.height
  );
}
```

Muret tekitas ka hiire tuvastus, nimelt *mouseover* sündmusele reageerimine. Reageerimine küll toimub, kuid võtab aega ning töötab vaid teatud nurga alt ruudule lähenedes. Mouseover käsklus töötas Internet Exploreris probleemivabalt.

3.1.2 KineticJS

Püstitatud probleemi lahendamine möödus suhteliselt probleemi vabal, lähtudes õpetustest ning dokumentatsioonist.

Võrreldes *JavaScript*'iga ei ole tarvis *KineticJS*'i puhul lõuendit puhastada, pole tarvis ruudu või hiire asukohta tuvastada ning ruudule on antud atribuudid, mis seisavad algse ning sündmuse järgse kuju eest ning lisaks ka ruudu positsioon, suurus, värv ning märge, et muutust ei ole toimunud.

Koodinäide 6. Ruudu ning atribuutide defineerimine.

```
var rectAttributes1 = {
  x: 0,
  y: 0,
  width: 100,
  height: 100,
  opacity: 1,
  fill: 'red',
  changed: false
};
```

Igale ruudule luuakse algne ja uus olek, mis muudetakse hiljem vastavas funktsioonis, millega lisatakse ruudule uued atribuudid.

Koodinäide 7. Ruudu algne ning sündmusejärgsete atribuutide määramine.

```
var rect1 = new Kinetic.Rect().setAttrs(rectAttributes1);
rect1.changed = false;
var defaultRect1 = new Kinetic.Rect().setAttrs(rectAttributes1);
```

Koodinäide 8. Funktsioon, ruudu atribuutide muutmiseks.

```
rect1.on('click', function() {
  if (this.changed) {
    this.setAttrs(defaultRect1.getAttrs());
    this.changed = false;
  } else {
    this.setAttrs({
      width: 50,
      height: 50
    });
    this.changed = true;
  }
  stage.add(layer);
});
```

3.1.3 FabricJS

Esiteks defineeriti autori poolt ruut, algsete atribuutidega ning identifikaatoriga. *FabricJS*'i üheks omapäraks on see, et objekti põhipunkt pole mitte vasak ülemine nurk vaid keskpunkt.

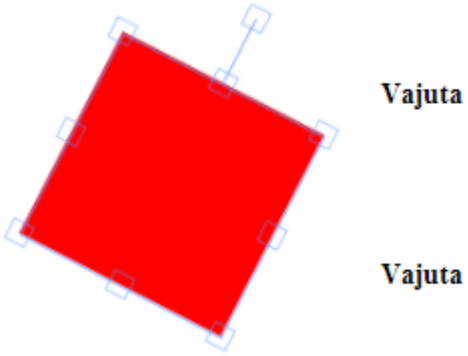
Koodinäide 9. Ruudu defineerimine,

```
var rect1 = new fabric.Rect({
  top: 50,
  left: 50,
  fill: 'red',
  width: 100,
```

```
  height: 100,
  id: 1
});
```

FabricJS käitus huvitavalt tänu oma vaikeseadetele, nimelt kõik objektid, kui neid pole just staatiliselt lõuendile loodud, on valitavad, mis võimaldab neid liigutada, kujundi suurust muuta ja kujundit keerata.

Ekraanitõmmis 3. Ruudu pööramine ning suuruse muutmine *FabricJS*'is.



Seega pole otseselt tarvis eraldi koodina seda funktsionaalsust lisada, kuna see on põhimõtteliselt *Flash*'ist tuttav graafiline interpoleerimine (*tween*), sündmusega määrata ruudu suuruse muutumine.
Problemaatiline oli antud teegi puhul see, et keerukas on panna objekte hiire sündmusi kuulama. Seega lisati igale ruudule identifikaator ning hiire sündmus lõuendile, kuid see reageerib siiski vaid juhul, kui konkreetsele objektile vajutatakse.

Koodinäide 10. Funktsioon, mis tuvastab hiire vajutuse ning määrab selle õigele ruudule.

```
canvas.on('mouse:down', function(e) {
  if (e.target.id === 3) {
    e.target.opacity = 0.5;
  }
});
```

Lisaks puudub *FabricJS*'il hiire liigutamise tuvastamise sündmus ehk *mousemove*.

3.1.4 SVGjs

Autor defineeris ruudud, lisades neile asukoha, suuruse ning värvi.

Koodinäide 11. Ruudu defineerimine.

```
var rect1 = draw.rect(100, 100).move(0, 0).fill('#f00')
```

Seejärel lisas igale ruudule funktsiooni, mis viis läbi muudetava sündmuse

Koodinäide 12. Funktsioon, millega muudetakse ruudu suurust.

```
rect1.click(function() {
  this.scale(0.5, 0.5)
})
```

SVGjs saab vaatamata oma kompaktsusele püstitatud ülesandega edukalt hakkama. Suurim segadusetekitaja on väga kokku surutud ja lihtne süntaks, mis oma lihtsusega tekitab segadust. Ebameeldivaim faktor on see, et ainukene hiiresündmus on *click*.
Probleemi valmistas veel ruudu suurendamine, nimelt kahekordistades ruudu suurust liikus see ühtlasi ka oma algsest asukohast diagonaalselt algse suuruse võrra ära, selle parandamiseks pidi objekti uuesti defineerima ning algsele kohale liigutama.

Koodinäide 13. Funktsioon ruudu suuruse muutmiseks ning ruudu algupärase asendi säilitamiseks.

```
rect2.click(function() {
  this.scale(2, 2)
  this.move(0, 110)
})
```

3.2 Pildi animeerimine

Autor impordib pildi lõuendile, seejärel defineerib vajalikud muutujad (algasend, lõppasend, lõuendi suurus, liikumise kiirus) ning seejärel loob funktsioonid, mis käivitavad liikumise. Olenevalt vajadusest luuakse üks või kaks funktsiooni, mis muudavad liikumise kahepoolseks, ehk algselt liigutatakse pilti vasakult paremale, kui pilt jõuab lõpp-punkti, liigutatakse pilt tagasi algpunkti.

Autor valis sellise ülesande, kuna animeerimise puhul on lihtsam importida pilt, kui luua samasugune tulemus kombineerides raamistiku- või teegipõhiseid kujundeid. Lisaks, kui pilt on imporditud on seda vaja ka animeerida.

Ekraanitõmmis 4. Liikumise algasend.

Ekraanitõmmis 5. Liikumise lõppasend.

3.2.1 JavaScript

Autor kasutas ülesande lahendamiseks *JavaScript*'i APIt *requestAnimationFrame* ning üht funktsiooni, mis tegeles animeerimisega.

Koodinäide 14. Funktsioon, mis liigutab animeerib pilti.

```
function right() {
  reqAnimFrame = window.mozRequestAnimationFrame ||
            window.webkitRequestAnimationFrame ||
            window.msRequestAnimationFrame ||
            window.oRequestAnimationFrame;
  reqAnimFrame(right);

  if (!reachedDestination) {
```

```
    x += speed;
  }
  if (reachedDestination) {
    x -= speed;
  }
  if (x == moveDistance) {
    reachedDestination = true;
  }
  if (x == 0) {
    reachedDestination = false;
  }
  draw();
}
```

Sama ülesanne oleks edukalt sooritatav ka traditsioonilisema lähenemisega, ehk kasutades taimereid ning luues kaks funktsiooni, mõlema liikumissuuna tarvis. Märkimisväärseid probleeme ning anomaaliaid selle ülesande puhul ei tekkinud.

3.2.2 KineticJS

Autor lõi ülesande lahendamiseks kaks funktsiooni, mõlemas suunas liikumise jaoks.

Koodinäide 15. Funktsioon, mis liigutab pilti paremas suunas.

```
function animateRight() {
  var animslide = new Kinetic.Animation(function(frame) {
    var speed = 5;
    layer.move(5, 0);
    if (layer.getAttr('x') == destination) {
      animateLeft();
    }
  }, layer);
  animslide.start();
  console.log('Liigub paremale');
```

}

Animatsioon liigub, kuid ühe huvitava anomaaliaga. Nimelt peale viiendat liikumist hakatakse animatsiooni kiirust liitma, mille tulemusena pilt lõpuks lõuendilt välja sõidab.

Ekraanitõmmis 6. Konsooli paneelist, milles kajastub *KineticJS*'i anomaalia.

Lisaks ei võimalda *KinetiJS* otsest pildi, kui objekti animeerimist. Esiteks lisatakse pilt *imageobject*ina, seejärel lisatakse see uuele kihile ning kiht omakorda lõuendile.

Koodinäide 16. Pildi faili atribuutide määramine ning kihile paigutamine.

```
imageObj.onload = function() {
  var person = new Kinetic.Image({
    x: 0,
    y: 0,
    image: imageObj,
    width: 200,
    height: 200
});
```

layer.add(person);

stage.add(layer);
};

3.2.3 FabricJS

Autor lõi kaks funktsiooni, millega määratakse liikumissuunad ning peatumine õigetel koordinaatidel.

Koodinäide 17. Funktsioon, mis animeerib pilti paremale poole.

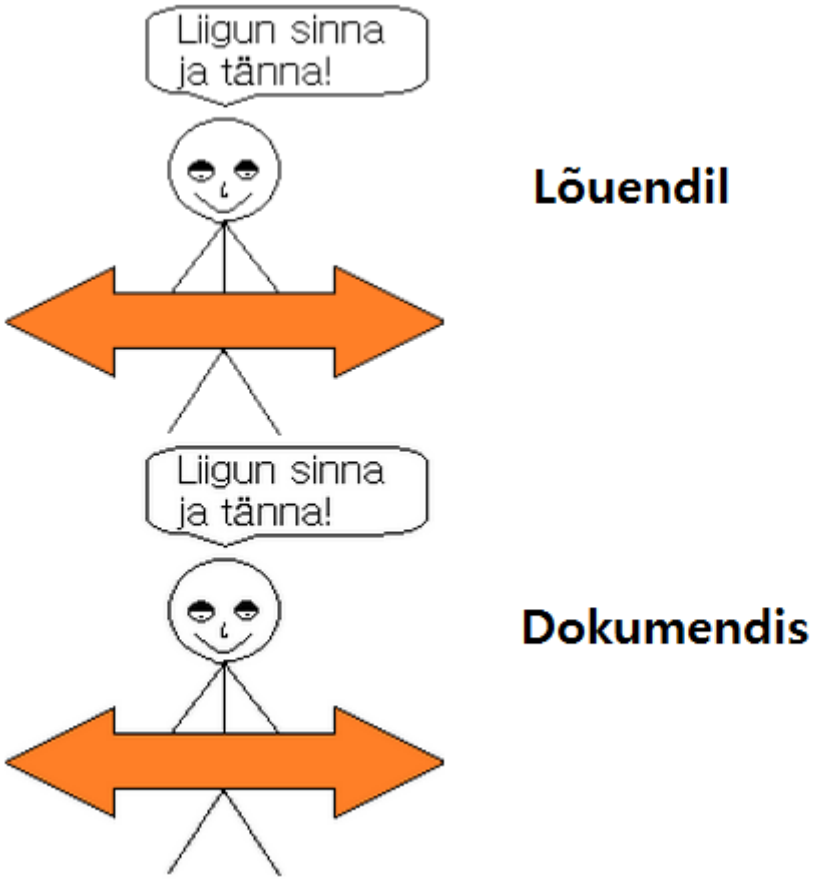
```
function animateRight() {
  console.log('right');
  imgInstance.animate('left', '+=400', {
    duration: 1000,
    onChange: canvas.renderAll.bind(canvas),
    onComplete: function() {
      animateLeft();
    }
  });
}
```

FabricJS võimaldab animeerida pildi objekti otse, kuid pildi objekti importimine on vigane. Kui pilti importida arvutis olevast kaustast, siis seda saab teha vaid läbi HTMLi, mitte otse läbi *FabricJS*'i, seega dubleeritakse pilti, üks läheb lõuendile objektiks ja teine jääb lihtsalt pildina lõuendi alla, nagu see originaalkujul imporditud oli, seega tuleb lõuendi all asuv pilt lihtsalt ära peita.

Koodinäide 18. Pildi faili importimine *FabricJS*'is.

```
<img src="images/persoon.png" id="person" style="visibility: hidden;">
```

Koodinäide 19. *FabricJS*'i pildi importimise viga.



3.2.4 SVGjs

SVGjs'iga oli antud ülesande lahendamine autori arvates kergeim. Kuna antud teek võimaldab väga lihtsat süntaksit on ülesanne koodi mahult eriti väike. Piisas vaid ühest funktsioonist, mis sisaldab endas tagasi kutset ning selle sama funktsiooni korduv välja kutsumine.

Koodinäide 20. Pildi animeerimine edasi-tagasi.

```
animatePerson();

function animatePerson() {
        image.animate(speed).move(destination, 0).after(function() {
                image.animate(speed).move(0, 0).after(function() {
                        animatePerson();
```

```
            });
        });
}
```

Animatsioon küll liigub veidi imelikult, kuna *move* käsklusele on teegi enda poolt juurde kirjutatud vaibuv liikumine, seega jääb mulje, nagu animatsiooni kiirus muutuks olenevalt positsioonist.

3.3 Autori soovitused

Käesolevas seminaritöös on käsitletud *JavaScript*'i puhtal kujul, raamistikku *KineticJS* ning teeke *FabricJS* ja *SVGjs.* Loodud näidisülesannetega said kõik töövahendid edukalt hakkama. Igal kasutajal on alati omad eelistused, millist töövahendit, mingi ülesande sooritamiseks kasutada, kuid siinkohal annaks Autor oma subjektiivse soovituse enda kogemustest vastavate töövahenditega.

Nimelt autori suurimaks eelistuseks on teek *FabricJS*, millega ei tekkinud ülesandeid lahendades peaaegu ühtki probleemi, lisaks on *FabricJS*'il suurim brauseri toetus ning töökindlus. Olenemata projekti suurusest on autori kindlaks soovituseks *FabricJS.*

Kokkuvõte

Käesolevas seminaritöö eesmärgiks oli anda ülevaade *JavaScript'*iga animeerimise võimalustest, tutvustada mõnda lõuendipõhist raamistikku ja teeki.

Autor andis ülevaate animeerimisest *JavaScript*'iga. Koostas kriteeriumid käsitletavate raamistike ja teekide valikuks, valis neile kriteeriumitele vastavalt teegid *SVGjs*, *FabricJS* ja raamistiku *KineticJS*, milledest andis lühiülevaate.

Autor koostas kaks näidisülesannet, millised realiseeris nii puhta JavaScripti, kui ka käsitletavate teekide ja raamistiku abil. Selle põhjal andis autor hinnangu käsitletud vahenditele. Parimaks *JavaScript*'i vahendks animatsioonide loomisel osutus teek *FabricJS*, mis toob käsitlusse objekti mõiste ja teeb animeerimise sellega lihtsamaks ja sarnasemaks mitmete muude vahenditega, sealhulgas ka *Adobe Flash*'iga.

Töö kirjutamise käigus loodud näidised on kättesaadavad aadressil www.tlu.ee/~heindrig/seminar

Käesolevat seminaritööd võiks kasutada õppematerjalina neile, kes vajavad animeerimise vahendeid veebiloomes.

Tabel 1. Veebisirvijate tugi.

	JavaScript	SVGjs	FabricJS	KineticJS
Chrome Versioon 30.0.1599.101 m	JAH	JAH	JAH	JAH
Firefox Versioon 24.0	**EI**	JAH	JAH	JAH
Explorer Versioon 10.0.9200.16721	JAH	JAH	JAH	JAH
Opera Versioon 11.52	**EI**	JAH	JAH	JAH
Android Versioon 2.3.6	JAH	**EI**	JAH	**EI**
Ios Versioon 7.0.3	Hiiretuvastus: JAH Liikumine: **EI**	JAH	JAH	Hiiretuvastus: **EI** Liikumine: JAH

Sõnastik

API (*Application Programming Interface*) – rakendusliides

Canvas – lõuend

DOM (*Document Object Model*) – dokumendiobjekti mudel

Framework – raamistik (on objektorienteeritud süsteemide puhul objektiklasside hulk, mis annab kasutajale või programmile omavahel seotud funktsioonide kollektsioon.)

GIF (*Graphics Interchange Format*) – Graafikavahetuse vorming

HTML (*Hypertext Markup Language*) – hüperteksti-märgistuskeel

JSON (*JavaScript Object Notation*) - JavaScripti objekti märkmed

Library – teek (valmiskompileeritud alamprogramm, mida põhiprogramm saab kasutada. Need alamprogrammid, mida kutsutakse ka mooduliteks, salvestatakse objektikoodidena.)

Plugin - pistikprogramm

Source code – lähtekood

SVG (*Scalable Vector Graphics*) – skaleeritav vektor graafika

Kasutatud kirjandus

Buck, S. (19. Oktoober 2012. a.). *The history of GIFs*. Kasutamise kuupäev: 23. oktoober 2013. a., allikas http://mashable.com/2012/10/19/animated-gif-history/

Chapman, S. (kuupäev puudub). *What is JavaScript Used For?* Kasutamise kuupäev: 23. oktoober 2013. a., allikas http://javascript.about.com/od/reference/a/javascriptpurpose.htm

Drowell, E. (kuupäev puudub). *Release Schedule*. Kasutamise kuupäev: 13. 10 2013. a., allikas Github: https://github.com/ericdrowell/KineticJS/wiki/Release-Schedule

Irish, P. (22. veebruar 2011. a.). *requestAnimationFrame for Smart Animating*. Kasutamise kuupäev: 23. oktoober 2013. a., allikas http://www.paulirish.com/2011/requestanimationframe-for-smart-animating/

Marinacci, J. (kuupäev puudub). *HTML Canvas: A Travelogue*. Kasutamise kuupäev: 23. oktoober 2013. a., allikas http://joshondesign.com/p/books/canvasdeepdive/chapter01.html#what_is_canvas

McBride, M. (26. september 2011. a.). *JavaScript canvas frameworks*. Kasutamise kuupäev: 23. oktoober 2013. a., allikas http://www.nublue.co.uk/blog/javascript-canvas-frameworks/

Rocheleau, J. (5. september 2012. a.). *A History Lesson on the Rise and Fall of Adobe Flash*. Kasutamise kuupäev: 23. oktoober 2013. a., allikas http://speckyboy.com/2012/09/05/a-history-lesson-on-the-rise-and-fall-of-adobe-flash/

Young, A. (24. mai 2010. a.). *History of JavaScript: Part 1*. Kasutamise kuupäev: 6. oktoober 2013. a., allikas http://dailyjs.com/2010/05/24/history-of-javascript-1/

Zim, J. (3. mai 2012. a.). *Sleek Animations with requestAnimationFrame*. Kasutamise kuupäev: 23. oktoober 2013. a., allikas http://www.joezimjs.com/javascript/sleek-animations-with-requestanimationframe/