دانشکده مهندسی کامپیوتر

ایجاد و توسعه یک بازی تحت وب به زبان جاواسکریپت با استفاده از قابلیت‌های HTML5

پایان‌نامه برای دریافت درجه کارشناسی

پروانه رحیمی

۸۷۵۲۱۱۵۴

استاد راهنما:

دکتر بهروز مینایی

دی ماه ۱۳۹۱

بسم الله الرحمن الرحيم

چکیده

HTML5 زبانی برای ارائه و نمایش محتوا در وب است که تکنولوژی اصلی اینترنت را شکل می‌دهد. این زبان، پنجمین نسخه استاندارد HTML است (که در سال ۱۹۹۰ ایجاد شد و در سال ۱۹۹۷ با عنوان HTML4 استاندارد شد.) که هنوز هم در حال توسعه است. هدف اصلی از ایجاد این زبان، بهبود زبان HTML به منظور پشتیبانی از آخرین تحولات مولتی‌مدیا و در عین حال خوانایی بهتر توسط انسان و فهم مداوم توسط کامپیوتر است.

HTML5 ویژگی‌های نحوی بسیاری به HTML افزوده است. از جمله این ویژگی‌ها عبارت است از: عناصر جدید <video>، <audio>، <canvas> و همچنین یکپارچه‌سازی محتوای گرافیک برداری مقیاس پذیر[۱]. این ویژگی‌ها به منظور آسان‌تر کردن قابلیت افزودن و مدیریت مولتی مدیا و محتوای گرافیکی در وب بدون نیاز به API[۲] و افزونه اختصاصی، طراحی شده‌اند.

اما عنصر مورد توجه در این پروژه canvas است. عنصر canvas تکنولوژی بسیار جدیدی به وب اضافه کرده که قابلیت ترسیم اشکال در سند وب را امکان‌پذیر ساخته است. قابلیت‌های این تکنولوژی را شاید بتوان با قابلیت‌های Flash مقایسه کرد. یعنی ترسیم اشکال، ایجاد انیمیشن و همچنین ایجاد و توسعه محیط‌های تعاملی از جمله بازی. در واقع، canvas عنصری در یک سند HTML است که دارای مشخصاتی از جمله طول و عرض می‌باشد که می‌توان از طریق جاواسکریپت با یک رابط کاربری به این ناحیه دسترسی پیدا کرد و در آن به ترسیم اشکال و ایجاد پویانمایی پرداخت، همچنین به دلیل استفاده از اسکریپت برای کار با عنصر canvas، امکان تعاملی کردن این محیط وجود دارد و می‌توان به طراحی بازی در آن پرداخت. شاید بتوان گفت، یکی از انگیزه‌های اساسی در ایجاد عنصر canvas، به وجود آوردن جایگزینی برای Flash بوده است تا بتوان علاوه بر مقابله با مشکلاتی که افزونه‌های Flash داشتند، امکانات آن‌ها را در محیط وب حفظ کرد.

---

[۱] Scalable Vector Graphics

[۲] Application Programming Interface

با توجه به اینکه عنصر canvas نسبتا جدید است و امکانات بسیار نویی در اختیار می‌گذارد، تصمیم به طراحی و پیاده‌سازی یک بازی دو بعدی تحت وب گرفته شد تا علاوه بر نشان دادن قابلیت‌های جدید HTML5، توانایی مرورگرهای جدید در پشتیبانی از این ویژگی‌ها نشان داده شود.

فهرست مطالب

فهرست شکل‌ها

فهرست جداول

مقدمه

HTML5 زبانی برای ارائه و نمایش محتوا در وب است که تکنولوژی اصلی اینترنت را شکل می‌دهد. این زبان، پنجمین نسخه استاندارد HTML است (که در سال ۱۹۹۰ ایجاد شد و در سال ۱۹۹۷ با عنوان HTML4 استاندارد شد.) که هنوز هم در حال توسعه است. هدف اصلی از ایجاد این زبان، بهبود زبان HTML به منظور پشتیبانی از آخرین تحولات مولتی مدیا و در عین حال خوانایی بهتر توسط انسان و فهم مداوم توسط کامپیوتر است.

HTML5 به گونه‌ای طراحی شده که در برگیرنده HTML4، XHTML و HTML Level2 DOM باشد.

مرورگرهای وب مستلزم پشتیبانی از این نسخه جدید HTML هستند تا توانایی نمایش صحیح صفحات وبی را که از HTML5 استفاده می‌کنند، داشته باشند. توسعه دهندگان مرورگرها باید نرم‌افزارهای خود را به روز کنند تا قابلیت استفاده از HTML5 را داشته باشند.

HTML5 یکی از گزینه‌های اصلی برای استفاده در برنامه‌های موبایل است. بسیاری از ویژگی‌های HTML5 با درنظر گرفتن قابلیت اجرا بر روی دستگاه‌های کم مصرفی همچون تلفن‌های هوشمند و تبلت‌ها، طراحی و توسعه داده شده است. بنا بر تحقیقات صورت گرفته در دسامبر ۲۰۱۱، پیش‌بینی‌ها حاکی از این است که تعداد دستگاه‌های موبایل سازگار با HTML5 در سال ۲۰۱۳ بیش از یک بیلیون خواهد بود.

HTML5 ویژگی‌های نحوی بسیاری به HTML افزوده است. از جمله این ویژگی‌ها عبارت است از: عناصر جدید <video>، <audio>، <canvas> و همچنین یکپارچه‌سازی محتوای گرافیک برداری مقیاس پذیر. این ویژگی‌ها به منظور آسان‌تر کردن قابلیت افزودن و مدیریت مولتی‌مدیا و محتوای گرافیکی در وب بدون نیاز به API و افزونه اختصاصی طراحی شده اند.

عناصر جدید دیگر، همچون <section>، <article>، <header> و <nav> برای ارتقای محتوای معنایی مستندات طراحی شده اند.

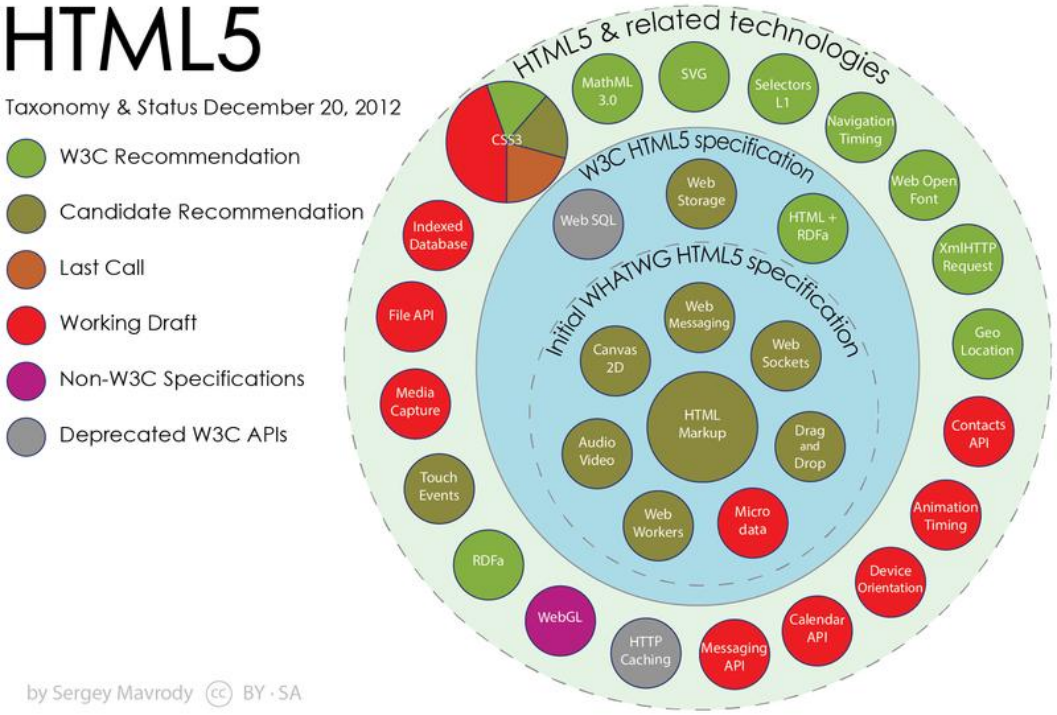
درحالی‌کهHTML5 با فلش مقایسه می‌شود، این دو تکنولوژی بسیار متفاوت‌اند. با این وجود، هر دو دارای قابلیت پخش صوت و ویدئو در صفحات وب هستند. همچنین استفاده از SVG و بردارهای گرافیکی در هر دو امکان‌پذیر است. HTML5 نمی‌تواند به تنهایی برای ایجاد انیمیشن و تعامل استفاده شود و باید به همراه CSS یا جاواسکریپت به کار برده شود.

HTML5 عناصر و صفات جدیدی را معرفی می‌کند که در وب‌سایت‌های مدرن کاربرد دارند. برخی از آنها، جایگزین معنایی بلوک عمومی <ivd> و عنصرخطی هستند. به عنوان نمونه، <nav>، <footer> یا <audio> و <video> به جای <object> استفاده می‌شوند. برخی ازعناصر HTML 4.01، از جمله عناصر نمایشی <font> و در این نسخه آورده نشده است. قواعد نحوی HTML5 به گونه‌ای طراحی شده است که با نسخه‌های قدیمی HTML سازگار باشد. همچنین این نسخه دارای خط مقدماتی جدید <html DOOCTYPE!> است.

HTML5 علاوه بر نشانه گذاری، APIهای جدیدی را معرفی می‌کند که می‌توانند همراه با جاواسکریپت استفاده شوند.

این APIها عبارتند از:

- عنصر canvas برای ترسیم دوبعدی
- پخش موسیقی به هنگام
- برنامه‌های وب آفلاین
- ویرایش مستند
- کشیدن و رها کردن نشانه‌گر
- مدیریت پیشینه مرورگر
- ذخیره‌سازی بر روی وب که رفتاری مشابه کوکیز دارد اما با گنجایشی بیشتر



شکل ۱– API **مربوط به** HTML5

تفاوت‌های HTML5 با HTML 4.01 و XHTML 1.X

- قواعد تجزیه[۳] جدید (تجزیه انعطاف پذیر، با سازگاری بیشتر)
- قابلیت استفاده از SVG خطی در text/html
- اضافه شدن عناصر جدید :

article, aside, audio, bdi, canvas, command, data,datalist, details, embed, figcaption, figure, footer,header, hgroup, keygen, mark, meter, nav, output, progress, rp, rt, ruby, section, source,summary, time, track, video, wbr

- اضافه شدن انواع جدید کنترل فرم:

dates and times, email, url, search, number, range, tel, color

- اضافه شدن صفات جدید :

charset (on meta), async (on script)

---

[۳] Parsing

- اضافه شدن صفات عمومی که می‌توانند برای هر عنصری به کار روند:

id, tabindex, hidden, data-* (custom data attributes)

- حذف شدن بسیاری از عناصری که موجب مشکل می شدند.

بسیاری از ویژگی‌های HTML5 در توسعه بازی کاربرد دارند که شاخص‌ترین آنها عنصر canvas است.

عنصر canvas جزیی از HTML است که امکان نمایش پویای تصاویر دو بعدی و bitmap را دارد.

canvas مدلی سطح پایین و رویه‌ای[۴] است که یک bitmap را به‌روز می‌کند و گراف صحنه‌ای تعریف شده‌ای درخود ندارد.

canvas شامل ناحیه‌ای با قابلیت ترسیم است که در HTML5 با دو صفت width و height تعریف می‌شود. کد جاواسکریپت از طریق مجموعه‌ای از توابع ترسیم، مشابه دیگر API دو بعدی متداول، به این ناحیه دسترسی دارد که در نتیجه امکان ایجاد گرافیک پویا را فراهم می‌سازد. از عنصر canvas در ایجاد گراف، انیمیشن، ترکیب تصاویر و بازی استفاده می‌شود.

نسخه‌های کنونی مرورگرهای Chrome، Firefox، IE، Safari، Opera و Konqueror قابلیت پشتیبانی از عنصر canvas را دارند. این در حالی است که نسخه‌های قدیمی تر IE (نسخه ۸ و قبل از آن)، canvas را پشتیبانی نمی‌کنند. اما، افزونه[۵]های Google و Mozilla برای پشتیبانی از این عنصر وجود دارند.

---

[۴] Procedural

[۵] PlugIn

قابلیت پشتیبانی مرورگرهای شناخته شده وب از canvas در سپتامبر ۲۰۱۲، در جدولی در زیر آورده شده است.

جدول ۱ـ مقایسه قابلیت مرورگرها در پشتیبانی از عنصر canvas

Internet Explorer	Firefox	Safari (Desktop)	Chrome	Opera (Desktop)	Safari (Mobile)	Opera (Mobile)	Android Browser
6.0	2.0- 6.0	3.1 - 3.2	4.0- 13.0	9.0 - 11.0	3.2	10.0	2.0
7.0	7.0	4.0	14.0	11.1	4.0	11.0	2.1
8.0	8.0	5.0	15.0	11.5	4.2 - 4.3	11.1	2.3,3.0
9.0	9.0	5.1	16.0	11.6	5.0	11.5	4.0
28.77%	19.70%	6.77%	30.01%	1.42%	2.79%	2.32%	3.02%

همان‌طور که گفته شد، عنصر canvas از ویژگی‌های جدید HTML5 است. این عنصر، قابلیت افزودن و مدیریت مولتی مدیا و محتوای گرافیکی در وب را بدون نیاز به API و افزونه اختصاصی، ممکن ساخته است. به همین جهت، در این پروژه، تصمیم به طراحی و پیاده‌سازی یک بازی دو بعدی تحت وب گرفته‌شد تا توانایی مرورگرهای جدید در پشتیبانی از این ویژگی‌های جدید نشان داده شود.

فصل اول

عنصر <canvas>

۱-۱- مقدمه

از آنجایی که هدف اصلی این پروژه ارزیابی امکانات جدید در HTML5 و به کار گیری آن‌ها در یک پیاده‌سازی جامع جهت استفاده از این امکانات در وب می باشد، لازم است که ابتدا به شرح کاملی از مهم‌ترین عنصر جدید در HTML یعنی canvas که بنای اصلی پیاده‌سازی این پروژه است و قابلیت ها و رابط کاربری مربوط به آن بپردازیم.

canvas که از طریق رابط کاربری خود یک محیط گرافیکی را در درون یک سند وب ایجاد می‌کند، عنصری بسیار نو در وب است که در استانداردهای کنسرسیوم وب تا زمان به وجود آمدن HTML5 نظیری نداشته است و تا زمان به وجود آمدن آن وظیفه انتقال محتوای چند رسانه‌ای عمدتا بر عهده افزونه‌های ثالث بوده است.

شیوه کار با canvas جدا از استاندارد خود HTML5 بوده و به کانتکستی که از رابط کاربری آن دریافت می‌شود برمی‌گردد، کانتکستی که در این پروژه از آن استفاده می‌شود و در این فصل به توضیح توابع رابط کاربری آن می‌پردازیم، 2D Context بوده که امکان ایجاد هر نوع تصویر دو بعدی برداری را به کاربر می‌دهد.

۱-۲- عنصر <canvas>

<canvas> عنصری از HTML5 است که معمولا از طریق کد جاواسکریپت برای ترسیم گرافیک به‌کار می‌رود. به عنوان مثال، از این عنصر می‌توان برای ترسیم گراف، ایجاد ترکیب‌بندی عکس و یا ایجاد انیمیشن استفاده کرد.

canvas اولین بار توسط شرکت اپل معرفی شد و بعدها در Google Chrome و Safari پیاده‌سازی شد.

در این فصل، چگونگی پیاده‌سازی <canvas> در صفحات HTML5 شرح داده خواهد شد.

برای استفاده از <canvas>، نیاز به درک پایه‌ای از HTML و جاواسکریپت است.

۱-۲-۱- خصوصیات عنصر <canvas>

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

این عنصر به عنصر<img> شباهت دارد، تنها تفاوت این است که دو ویژگی scr و alt را ندارد. عنصر <canvas> فقط دو صفت width و height دارد که هر دو اختیاری هستند و می‌توانند با استفاده از خاصیت DOM تعیین شوند. زمانی‌که این دو مقداردهی نشوند، canvas با طول ۲۰۰ پیکسل و عرض ۱۵۰ پیکسل مقدار دهی می‌شود.

صفت id مختص عنصر <canvas> نیست، بلکه یکی از صفات پیش‌فرض HTML است که می‌تواند بر روی همه عناصر HTML اعمال شود. بهتراست که همواره از یک id استفاده شود تا شناسایی آن در کد جاواسکریپت راحت‌تر شود.

عنصر <canvas> می‌تواند مانند هر تصویر دیگری دارای سبک باشد (حاشیه ، زمینه و غیره). البته این قواعد برعمل ترسیم روی canvas تاثیری نخواهند داشت. اگر هیچ قاعده‌ای روی canvas به کار نرود، canvas در ابتدا شفاف خواهد بود.

۱-۲-۲- محتوای جایگزین

از آن جایی که عنصر <canvas> به نسبت، عنصر جدیدی است و در برخی مرورگرها پیاده‌سازی نشده است (به عنوان مثال در نسخه‌های پایین تر از IE۹). به همین جهت، نیازمند روشی برای ایجاد یک محتوای جایگزین در زمان‌هایی که مرورگر قابلیت پشتیبانی از عنصر <canvas> را ندارد، هستیم.

با قرار دادن یک محتوای جایگزین[۶] درون عنصر <canvas>، مرورگرهایی که عنصر <canvas> را پشتیبانی نمی کنند، به محتوای جایگزین درون <canvas> مراجعه می‌کنند.

برای نمونه، به عنوان جایگزین عنصر <canvas> می‌توان توصیفی متنی و یا تصویری استاتیک به کار برد، که در زیر نمونه‌ای از آن را می‌توان مشاهده کرد.

```
<canvas id="stockGraph" width="150" height="150">
 current stock price: $3.15 +0.15
</canvas>

<canvas id="clock" width="150" height="150">
 <img src="images/clock.png" width="150" height="150" alt=""/>
</canvas>
```

۱-۲-۳- الزام استفاده از برچسب </canvas>

در پیاده‌سازی Safari توسط شرکت اپل، عنصر <canvas> تقریبا با همان روش عنصر <img> پیاده‌سازی شده است، به همین دلیل فاقد برچسب پایانی است. اما، به دلیل کاربرد گسترده عنصر <canvas> در وب، امکاناتی برای جایگزینی محتوا باید در نظر گرفته شود، به همین دلیل پیاده‌سازی Mozilla الزاما شامل برچسب پایانی است.

اگر نیاز به استفاده از محتوای جایگزین نباشد، عبارت زیر

```
<canvas id="foo" ...></canvas>
```

با Safari و Mozilla هر دو سازگار است، با این تفاوت که Safari برچسب نهایی را نادیده می‌گیرد.

---

[۶] FallBack Content

۱-۲-۴- بررسی پشتیبانی مرورگر از عنصر <canvas>

اشاره شد که محتوای جایگزین در مرورگرهایی که عنصر <canvas> را پشتیبانی نمی کنند، نمایش داده می‌شود. با این وجود، کد اسکریپت هم می‌تواند پشتیبانی و یا عدم پشتیبانی را در زمان اجرا بررسی کند. این کار به راحتی به صورت زیر انجام می‌شود.

```
var  canvas = document.getElementById('tutorial');
if  (canvas.getContext){
  var  ctx = canvas.getContext('2d');
  // drawing code here
} else  {
  // canvas-unsupported code here
}
```

۱-۳- ترسیم اشکال در canvas

برخلاف SVG، عنصر <canvas> فقط یک شکل اصلی را پشتیبانی می‌کند و آن مستطیل است. تمام اشکال دیگر از طریق ترکیب یک یا چند مسیر ایجاد می‌شوند. مجموعه‌ای از توابع ترسیم مسیر وجود دارد که با استفاده از آنها، می‌توان اشکال بسیار پیچیده را ایجاد کرد.

۱-۳-۱- مستطیل

برای رسم مستطیل بر روی canvas از سه تابع می‌توان استفاده کرد.

```
fillRect(x,y,width,height)
```

تابع بالا برای ترسیم مستطیل توپر به کار می‌رود.

```
strokeRect(x,y,width,height)
```

تابع بالا برای ترسیم مستطیل تو خالی به کار می‌رود.

```
clearRect(x,y,width,height)
```

این تابع، ناحیه مورد نظر را پاک می‌کند و مستطیلی کاملا شفاف رسم می‌کند.

این توابع، هر سه پارامترهای یکسانی می‌گیرند. x وy موقعیت را روی canvas نسبت به گوشه‌ی بالای سمت چپ مستطیل نشان می‌دهند.

۱-۳-۲- ترسیم مسیر

به منظور رسم اشکال با استفاده از مسیر، گام‌های دیگری نیز باید به صورت زیر طی شود:

- beginPath()
- closePath()
- stroke()
- fill()

اولین گام در ایجاد مسیر، فراخوانی تابع beginPath است. مسیرها به صورت لیستی از مسیرهای کوچکتر(خط، منحنی و غیره) ذخیره می‌شوند که با هم شکلی را می‌سازند. هر بار که این تابع فراخوانی می‌شود، لیست مجددا تنظیم می‌شود و می‌توان اشکال جدید رسم کرد.

گام دوم، فراخوانی توابعی است که مسیرها را برای رسم مشخص می‌سازند.

سومین گام که فراخوانی آن اختیاری است، تابع closePath است که از طریق رسم خطی مستقیم از نقطه کنونی به نقطه اولیه، شکل را به مسیری بسته تبدیل می‌کند. اگر مسیر قبل از فراخوانی این تابع، مسیری بسته باشد و یا مسیر تنها از یک نقطه تشکیل شده باشد، این تابع کاری انجام نمی‌دهد.

گام نهایی فراخوانی تابع stroke یا fill است. فراخوانی هریک از این دو، شکلی را بر روی canvas رسم می‌کند. stroke برای رسم شکل با خط دور و fill برای رسم شکل بدون خط دور به کار می‌رود.

برای مثال کد زیر یک مثلث رسم می‌کند:

```
ctx.beginPath();
ctx.moveTo(75,50);
ctx.lineTo(100,75);
ctx.lineTo(100,25);
ctx.fill();
```

۱-۳-۳- تابع moveTo

تابعی بسیار کارامد که در واقع چیزی رسم نمی کند اما جزیی از لیست مسیرها است، تابع moveTo است. این تابع به مانند بلند کردن مداد یا خود کار از یک نقطه روی کاغذ و منتقل کردن آن به نقطه دیگری است.

```
moveTo(x,y)
```

تابع moveTo دارای دو آرگومان x و y است که مختصات نقطه جدید را مشخص می‌کنند.

زمانی که canvas مقدار دهی اولیه می‌شود و یا تابع beginPath فراخوانی می‌شود، برای قرار دادن نقطه شروع در جای دیگری از canvas نیاز به فراخوانی این تابع است. همچنین می‌توان از این تابع برای رسم اشکال غیر متصل استفاده کرد.

۱-۳-۴- خطوط

برای رسم خطوط مستقیم از تابع lineTo استفاده می‌شود.

```
lineTo(x, y)
```

این تابع دو آرگومان x وy می‌گیرد که مختصات نقطه انتهایی خط است. نقطه شروع بستگی به مسیر رسم شده قبلی دارد، به گونه‌ای که نقطه انتهایی مسیر قبلی به عنوان نقطه شروع خط بعدی تلقی می‌شود. نقطه شروع با استفاده از تابع moveTo قابل تغییر است.

۱-۳-۵- منحنی

برای رسم منحنی و دایره می‌توان از تابع arc استفاده کرد.

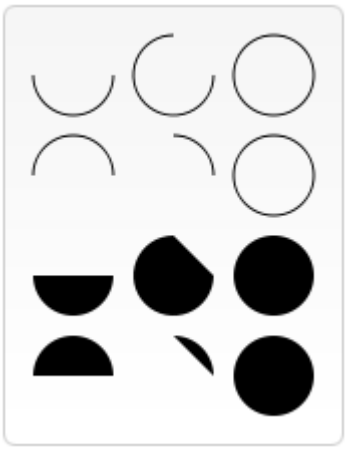
```
arc(x,y,radius,startAngle,endAngle,anticlockwise)
```

این تابع دارای پنج پارامتر است: x و y بیانگر مختصات مرکز دایره هستند. radius مشخص کننده شعاع دایره، پارامترهای startAngle و endAngle شروع و پایان منحنی را به رادیان مشخص می‌کنند. همچنین، زوایای آغازی و پایانی نسبت به محورx اندازه گرفته می‌شوند.

پارامتر anticlockwise متغیری boolean است، اگر دارای مقدار true باشد در جهت خلاف گردش عقربه‌های ساعت منحنی را رسم می‌کند، اما اگر false باشد، در جهت حرکت عقربه‌های ساعت منحنی رسم می‌شود.

برای نمونه، در مثال زیر ۱۲ منحنی مختلف با زوایای متفاوت رسم شده است، که تصویر حاصل از اجرای کد، در پایین آمده است.

```
for(var i=0;i<4;i++){
  for(var j=0;j<3;j++){
    ctx.beginPath();
    var x              = 25+j*50;       // x coordinate
    var y              = 25+i*50;       // y coordinate
    var radius         = 20;            // Arc radius
    var startAngle     = 0;             // Starting point on circle
    var endAngle       = Math.PI+(Math.PI*j)/2; // End point on circle
    var anticlockwise  = i%2==0 ? false : true; // clockwise or anticlockwise
    ctx.arc(x,y,radius,startAngle,endAngle, anticlockwise);
    if (i>1){
      ctx.fill();
    } else {
      ctx.stroke();
    }
  }
}
```



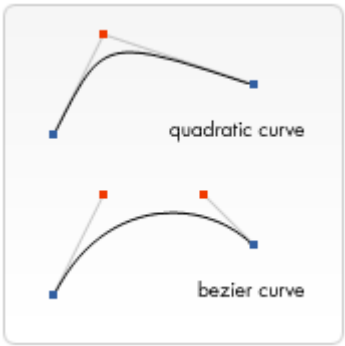
شکل ۲- پیاده‌سازی ۱۲ منحنی مختلف در canvas

۱-۳-۶- منحنی‌های bezier و quatratic

نوع دیگری از مسیرهای موجود، منحنی‌های bezier است که در دو صورت درجه دو و درجه سه وجود دارد. این دو عموما برای رسم اشکال پیچیده به کار می‌روند.

```
quadraticCurveTo(cp1x, cp1y, x, y)

bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
```

تفاوت این دو منحنی در شکل زیر به خوبی نشان داده شده است. منحنی quatratic درجه دوم، دارای دو نقطه آغازی و پایانی (نقاط مشخص شده با رنگ آبی) و یک نقطه کنترلی (مشخص شده با رنگ قرمز) است. درحالی‌که منحنی bezier درجه سه دارای دو نقطه کنترلی است.



شکل ۳- تفاوت دو منحنی bezier درجه دوم و درجه سوم

پارامترهای x و y در این توابع، مختصات نقطه انتهایی منحنی است. cp1x و cp1y مختصات نقطه کنترلی اول، cp2x و cp2y مختصات نقطه کنترلی دوم است.

استفاده از منحنی‌های bezier بسیار مشکل است، زیرا برخلاف نرم‌افزارهای ترسیم، بازخورد تصویری مستقیم برای دیدن عمل انجام شده وجود ندارد.

۱-۳-۷- مستطیل ها

علاوه بر سه تابعی که قبلا برای رسم مستقیم مستطیل روی canvas معرفی شد، تابع دیگری به نام rect وجود دارد که فقط مسیری مستطیلی شکل به لیست مسیرها می‌افزاید.

```
rect(x, y, width, height)
```

این تابع دارای چهار آرگومان است. x و y مختصات گوشه ی بالا، سمت راست مسیر مستطیلی جدید را مشخص می‌کنند. width و height طول و عرض مستطیل را مشخص می‌کنند.

هنگامی که این تابع اجرا می‌شود، تابع moveTo به طور خودکار با پارامتر (۰و۰) فراخوانی می‌شود. (نقطه شروع را به نقطه پیش فرض منتقل می‌کند.)

۱-۳-۸- استفاده از تصویر

یکی از ویژگی‌های جذاب canvas امکان استفاده از تصاویر است. این ویژگی این امکان را می‌دهد که بتوان به طور پویا ترکیب بندی عکس انجام داد. تصاویر می‌توانند با هر یک از فرمت‌های پشتیبانی شده (به عنوان مثال PNG،GIF و JPEG) مورد استفاده قرارگیرند.

وارد[۷]کردن تصویر عموما یک فرایند دو مرحله‌ای است:

۱ـ ابتدا ارجاعی به یک شیء تصویر جاواسکریپت یا عنصر دیگری از canvas نیاز است. این امکان وجود ندارد که مستقیما از یک URL یا مسیر استفاده کرد.

۲ـ سپس تصویر را با استفاده از تابع drawImage بر روی canvas رسم می‌کنیم.

---

[۷] Import

می توان شی جدید تصویر را به صورت زیر در اسکریپت ایجاد کرد.

```
var img = new Image();   // Create new img element
img.src = 'myImage.png'; // Set source path
```

زمانی که این اسکریپت اجرا شود، تصویرشروع به بارگذاری[۸] می‌کند. اگر تابع drawImage قبل از بارگذاری تصویر فراخوانی شود، با خطا مواجه خواهیم شد. به همین دلیل نیاز به استفاده از کنترل رخداد[۹] onload است.

```
var img = new Image();   // Create new img element
img.onload = function(){
  // execute drawImage statements here
};
img.src = 'myImage.png'; // Set source path
```

زمانی که ارجاع به شی تصویر آماده شد، می‌توان از تابع drawImage برای رسم تصویر بر روی canvas استفاده کرد. drawImage دارای سه شکل متفاوت است که در زیر قابل مشاهده است.

```
drawImage(image, x, y)
```

image ارجاعی به تصویر مورد نظر و x و y مختصات موقعیتی است که تصویر در آنجا قرار می‌گیرد.

دومین شکل تابع drawImage، دو پارامتر اضافه بر حالت قبل دارد و این امکان را می‌دهد که تصاویر با تناسبات متفاوت را بر روی canvas قرار داد.

```
drawImage(image, x, y, width, height)
```

---

[۸] Load

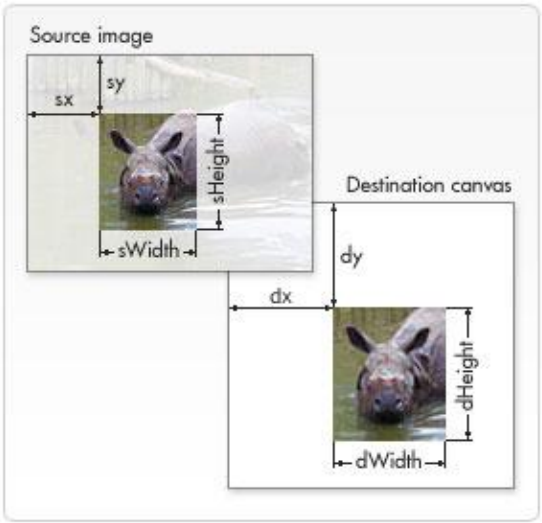
[۹] Event handler

x و y ابعاد تصویر را مشخص می‌کنند.

شکل سوم و آخر تابع drawImage دارای هشت پارامتر جدید است. این تابع را می‌توان برای برش قسمت هایی از تصویر مرجع و رسم آنها بر canvas به کار برد.

```
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)
```

اولین پارامتر image است که همان‌گونه که قبلا گفته شد ارجاعی به شی تصویر یا عنصر دیگری از canvas است. برای درک هشت پارامتر دیگر بهتر است به تصویر زیر مراجعه کرد. چهار پارامتر اول موقعیت و ابعاد تصویر برش داده شده را بر تصویر اصلی مشخص می‌کنند. چهار پارامتر بعدی، موقعیت و ابعاد را بر روی canvas هدف مشخص می‌کنند.



شکل ۴- هشت پارامتر تابع drawImage

برش ابزار کارآمدی برای ایجاد ترکیب‌بندی‌های متفاوت است. برای مثال، به منظور ایجاد یک نمودار، می‌توان تمام متن لازم را در یک تصویر PNG تنها در یک فایل ذخیره کرد و متناسب با نوع داده‌ها، نمودار را به‌راحتی برش داد. مزیت دیگر روش برش، عدم نیاز به بارگذاری تمام تصاویر است.

۱-۳-۹- تغییر شکل در canvas

استفاده از دو تابع زیر به هنگام ترسیم طرح‌های پیچیده ضروری است :

```
save()
restore()
```

این دو تابع برای ذخیره و بازیابی وضعیت و حالت canvas به‌کارمی‌روند. حالت ترسیم canvas، تصویری لحظه‌ای از تمام سبک‌ها و تغییر شکل‌های به کار برده شده است. این دو تابع، فاقد پارامتر هستند.

حالات canvas در یک پشته ذخیره می‌شود. هر زمان که تابع save فراخوانی می‌شود، حالت ترسیم کنونی بر روی پشته ذخیره می‌شود. حالت ترسیم شامل موارد زیر است:

- تغییرشکل‌های به کار برده شده (یعنی translate، rotate و scale)
- مقادیر ویژگی‌های :

strokeStyle, fillStyle, globalAlpha, lineWidth, lineCap, lineJoin, miterLimit, shadowOffsetX, shadowOffsetY,shadowBlur,shadowColor, globalCompositeOperation

- مسیر برش[۱۰] فعلی

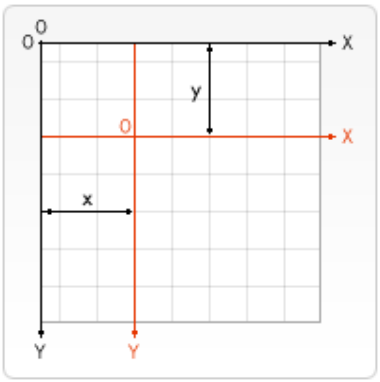
می توان تابع save را به تعداد دلخواه فراخوانی کرد.

با هر بار فراخوانی تابع restore، آخرین حالت ذخیره شده، از پشته بازگردانده می‌شود.

تابع translate به منظور انتقال canvas و نقطه مبدا به نقطه‌ای دیگر به کار برده می‌شود.

```
translate(x, y)
```

[۱۰] Clipping Mask



شکل ۵- استفاده از تابع translate به منظور جابه جایی نقطه مبدا

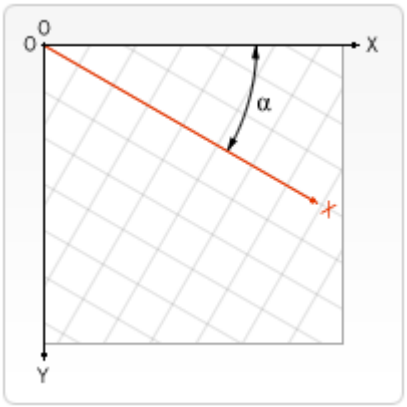
این تابع دارای دو آرگومان است. x مقدار جابه‌جایی canvas به چپ یا راست را تعیین می‌کند و y مقداری که canvas در راستای بالا یا پایین جابه‌جا می‌شود را مشخص می‌کند.

بهتر است قبل از هرگونه تغییر شکلی حالت canvas ذخیره شود. در بیشتر مواقع فراخوانی تابع translate راحت‌تر از انجام یک جابه‌جایی در جهت عکس، برای بازگشت به حالت اولیه است.

دومین تابع تغییر شکل، تابع rotate است. این تابع به منظور چرخش canvas حول نقطه مبدا کنونی به کار می‌رود.

```
rotate(angle)
```

تابع rotate فقط یک پارامتر دارد که نشان‌دهنده مقدار چرخش canvas است. چرخش در جهت حرکت عقربه‌های ساعت و بر حسب رادیان صورت می‌گیرد. (در شکل زیر نحوه چرخش نشان داده شده است.)



شکل ۶- جهت چرخش در تابع rotate

مرکز چرخش همواره نقطه مبدا canvas است. برای تغییر مرکز چرخش، می‌توان canvas را با استفاده از تابع translate جابه جا کرد.

تابع دیگر مورد استفاده در تغییرشکل، تابع scale است. این تابع برای کاهش یا افزایش واحد در canvas استفاده می‌شود. تابع scale در کوچک کردن یا بزرگ کردن اشکال یا تصاویر bitmap به کار برده می‌شود.

```
scale(x, y)
```

این تابع دارای دو پارامتر است. x ضریب مقیاس در جهت افقی و y ضریب قیاس در جهت عمودی است. هر دو پارامتر اعداد حقیقی و نه لزوما مثبت اند. مقادیر کوچک‌تر از یک، ابعاد واحد را کاهش می‌دهند و مقادیر بزرگ‌تر از یک، ابعاد واحد را افزایش می‌دهند. مقداردهی ضریب مقیاس با عدد یک، تاثیری بر ابعاد واحد نخواهد داشت. با انتخاب اعداد منفی می‌توان جهت محورها را عکس کرد.

به طور پیش فرض، یک واحد در canvas برابر دقیقا یک پیکسل است.

۵ آخرین تابع تغییرشکل، تابع transform است که امکان اعمال تغییرات بر ماتریس تبدیل را مستقیما فراهم می‌آورد.

```
transform(m11, m12, m21, m22, dx, dy)
```

این تابع، ماتریس تبدیل فعلی را در ماتریس تبدیلی که به صورت زیر تعریف می‌شود، ضرب می‌کند.

$$\begin{bmatrix} m11 & m21 \\ m12 & m22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

m11	m21	dx
m12	m22	dy
0	0	1

setTransform(m11, m12, m21, m22, dx, dy)

این تابع، ماتریس تبدیل فعلی را دوباره تنظیم می‌کند. سپس تابع transform را با همان آرگومان‌ها فراخوانی می‌کند.

۱-۴- انیمیشن

از آن جایی که از اسکریپت برای کنترل عناصرcanvas استفاده می‌شود، ایجاد انیمیشن‌های تعاملی امکان‌پذیر است. اما، از آن‌جایی‌که عنصر canvas برای ایجاد انیمیشن طراحی نشده است، (برخلاف نرم افزار فلش) محدودیت هایی در این رابطه، وجود دارد.

بزرگترین محدودیت این است که پس از انجام عمل ترسیم، شکل به همان صورت باقی می‌ماند. اگر نیاز به جابه جایی شکل مورد نظر باشد، نیاز است شکل و هر چیزی که قبل از آن رسم شده را مجددا ترسیم کرد. اما ترسیم مجدد فریم‌ها، زمان زیادی می‌برد و کارایی، به شدت وابسته به سرعت کامپیوتری که آن را اجرا می‌کند، خواهد داشت.

۱-۴-۱- گام‌های ایجاد یک انیمیشن ساده

گام‌های زیر برای برای رسم یک فریم باید طی شوند.

۱ـ پاک کردن canvas

به جز زمانی‌که اشکال ترسیم شده تمام فضای canvas را پر می‌کنند، در باقی موارد، باید تمام اشکالی که قبلا رسم شده‌اند، پاک شوند. آسان‌ترین شیوه برای انجام این کار استفاده از تابع clearRect است.

۲ـ ذخیره حالت وضعیت canvas

در صورت ایجاد تغییر در تنظیماتی که حالت canvas را تحت تاثیر قرار می‌دهند (مانند تغییر سبک‌ها و تغییر اشکال)، برای اطمینان از کاربرد حالت اولیه در زمان رسم هر فریم، باید حالت اولیه را ذخیره کرد.

۳ـ رسم اشکال متحرک

مشخص کردن اتفاقات مربوط به هر فریم، در این مرحله انجام می‌شود.

۴ـ بازیابی حالت canvas

اگر وضعیت canvas ذخیره شده باشد، باید قبل از رسم فریم جدید بازیابی شود.

۱-۴-۲- کنترل انیمیشن

اشکال با به کار بردن مستقیم توابع canvas یا فراخوانی توابع متداول، رسم می‌شوند. در شرایط معمولی بعد از پایان اجرای اسکریپت، نتایج قابل مشاهده‌اند. به عنوان مثال، ایجاد انیمیشن در درون حلقه for امکان‌پذیر نیست. به همین دلیل به شیوه‌ای دیگر برای اجرای تابع ترسیم بعد از مدت زمانی معین، نیاز است. دو روش برای کنترل انیمیشن وجود دارد.

اولین روش استفاده از دو تابع setInterval و setTimeout است که می‌توانند برای فراخوانی تابعی در دوره‌های زمانی مشخص استفاده شوند.

```
setInterval(animateShape,500);
setTimeout(animateShape,500);
```

اگر نیاز به تعامل با کاربر نباشد، بهترین شیوه استفاده از تابع setInterval است که به طور مکرر اجرا می‌شود. در مثال بالا تابع animateShape هر ۵۰۰ میلی ثانیه اجرا می‌شود. تابع setTimeout فقط یک بار بعد از گذشت زمان مشخص، اجرا می‌شود.

روش دوم برای کنترل انیمیشن، ورودی کاربر است. در طراحی بازی، برای کنترل انیمیشن می‌توان از رخدادهای صفحه کلید یا نشانه‌گر استفاده کرد. با مشخص کردن eventListner می‌توان تعاملات کاربر را به دست آورد و توابع انیمیشن را اجرا کرد.

۱-۵- جمع‌بندی و نتیجه‌گیری

در این فصل، عنصر canvas و نحوه تعامل با آن از طریق جاواسکریپت، به تفصیل شرح داده شد و دیده شد که قابلیت های این عنصر به‌حدی است که می تواند در ایجاد و توسعه بازی‌های تحت وب به کار رود و شاید حتی بتوان آن را جایگزینی برای Flash دانست. به همین دلیل، تصمیم به طراحی و توسعه یک بازی دوبعدی با استفاده از عنصر canvas گرفته شد تا بتوان به طور عملی توانایی‌های این عنصر در ایجاد انیمیشن و ایجاد تعامل نشان داده شود. در فصل سوم، شرح طراحی بازی در قالب مستندی آورده شده است و درفصل چهارم جزئیات پیاده‌سازی بازی طراحی‌شده، در قالب کد جاواسکریپت شرح داده شده است.

فصل دوم

مستند طراحی بازی

BAMIDELE

۳-۱- مقدمه

در این پروژه علاوه بر اینکه تلاش شد تا امکانات جدید HTML به کار گرفته شود تا قابلیت‌های آن در پیاده‌سازی مشخص شوند، طرح و ایجاد یک برنامه واقعی نیز مدنظر بوده است. این موضوع از آن باب است که نشان داده شود چطور کارکرد های اصلی محتواهای چندرسانه‌ای در دنیای وب، می توانند در قالب این امکانات نیز ایجاد شوند.

احتمالا مهم‌ترین این کارکردها برنامه‌های تعاملی به خصوص بازی‌ها می‌باشند، از این رو در این پروژه نیز سعی شد تا یک بازی واقعی طرح ریزی و پیاده شود و از آن جایی که این بازی می‌بایست با کیفیت‌های بازی‌های موجود در وب قابل رقابت باشد، نیاز به طرحی قوی و محکم داشت. در این فصل مستند طرح این بازی مطرح می شود و به توضیح قسمت‌های مختلف آن می‌پردازیم.

۳-۲- انتخاب نام بازی

بامی‌دله (BAMIDELE) نامی آفریقایی است، به معنی "مرا تا خانه دنبال کن " که با توجه به نوع و داستان بازی انتخاب شده است.

۳-۳- شرح داستان بازی

ایده‌ی اصلی بازی براساس فرهنگ بومی مردم آفریقا شکل گرفته است. بدین منظور از دو المان بارز هنر آفریقا، یعنی موسیقی و صنایع دستی در طراحی بازی استفاده شده است. صنایع دستی بارز این منطقه که در آشنایی با فرهنگ مردم ساکن این منطقه اثر به سزایی دارد، ماسک‌ها و سبدهایی است که توسط مردم آفریقا ساخته می‌شود. علاوه بر این، موسیقی بومی این منطقه را عموما پرکاشن شکل می‌دهد. به همین جهت، موسیقی متن بازی توسط سازهای پرکاشن اجرا می‌شود و در جای دیگری از بازی، ساز تامبورین که باز هم نمونه‌ای از موسیقی آفریقایی است، به عنوان عنصر تشویق‌کننده بازیکن دیده می‌شود.

عنصر بارز دیگری که در هنر آفریقایی به وفور دیده می‌شود، طیف گسترده‌ی رنگ‌ها است، به همین دلیل در طراحی این بازی سعی در استفاده از طیف رنگ‌های آفریقایی شده است.

عاملی که کنترل آن توسط فرد بازیکن صورت می‌گیرد و عامل اصلی بازی است، خطی با قابلیت حرکت منحنی‌وار است که در طراحی آن، این نکته در نظر گرفته شده که الهام گرفته از طبیعت و نمادی از حرکت و پویایی باشد. خط، در محیطی که عناصر فرهنگ آفریقایی حضور دارند در حرکت است و برخورد این خط با عناصر متفاوت، با توجه به نوع عنصر، اثر مثبت یا منفی برنحوه بازی شخص می‌گذارد و در پایان با توجه به نحوه بازی شخص به او امتیاز تعلق خواهد گرفت.

۳-۴- کاراکترها

کاراکتر اصلی بازی در قالب یک خط است که کنترل آن توسط شخص بازیکن صورت می‌گیرد. دیگر کاراکترهای موثر بر بازی عبارت است از: موسیقی، سبدهایی با اثرات متفاوت بر نحوه بازی شخص، ماسک هایی با توانایی حمله و عوامل تشویق کننده به شکل دست.

۳-۵- طراحی مراحل بازی

در این بازی، مراحل به گونه‌ای طراحی شده که در هر مرحله، کارکرد نوعی از سبدها که بر بازی شخص اثر منفی دارد، تغییر خواهدکرد و از خود کارکردی جدید و اضافه بر کارکردهای قبلی نشان خواهد داد. به عنوان نمونه، ماسک هایی که توانایی حمله به خط را دارند به بازی اضافه خواهند شد، فرد بازیکن می‌تواند واکنشی دفاعی در برابر عناصری که قصد حمله دارند، انجام دهد و در صورتی که از خود دفاع نکند، ماسک‌ها به او حمله می‌کنند. هر کدام از این کارکردها به تدریج به بازی افزوده خواهد شد. در نهایت با در نظر گرفتن همه این عوامل، تصمیم گرفته شد که بازی در دو مرحله پیاده‌سازی شود.

۳-۶- روند بازی

بازی با نواختن موسیقی شروع می‌شود و با اتمام موسیقی به پایان خواهد رسید (به منظور درگیر کردن بازیکن با موسیقی). با شروع مرحله اول بازی، کنترل خط توسط شخص بازیکن آغاز می‌شود. بازیکن می‌تواند با برخورد با سبدهایی که به کسب امتیاز کمک می‌کنند، امتیاز خود را افزایش دهد. این سبدها دارای طیف رنگی متفاوت و حرکت چرخان و دایره‌ای شکل هستند اما، در صورت برخورد با عناصر منفی بازی، که شامل سبد هایی با طیف رنگی و نوع حرکت متفاوت است، از امتیاز او کاسته خواهد شد. برخورد با این نوع سبد، علاوه بر کاهش امتیاز، بر شکل خط نیز تاثیر می‌گذارد و نوعی حرکت موجی و لرزش در خط پدید می‌آورد. در طراحی این نوع حرکت سعی بر آن شده که حس ضعف و اعمال ضربه بر خط به بازیکن القا شود.

از جمله نکاتی که در طراحی نحوه حضور سبدها در صفحه به آن توجه شده است، عبارت است از: حضور اتفاقی انواع سبد در هر لحظه در صفحه، تعداد سبدها و نسبت حضور هر یک از انواع سبد در صفحه و سرعت حرکت رو به جلو سبدها در صفحه. در اندازه سبدها نیز سعی بر آن بوده است که طیفی از ابعاد، از هر یک از انواع سبدها در بازی وجود داشته باشد.

با پایان مرحله اول و شروع مرحله دوم، دست هایی که تامبورین می‌نوازند به منظور تشویق بازیکن وارد صفحه می‌شوند و بعد از مدت زمانی مشخص از صفحه خارج می‌شوند. در این مرحله، علاوه بر سبدهای مرحله اول، نوع دیگری از سبدها با کارکردی متفاوت به بازی اضافه می‌شود. سبدهایی که در این مرحله اضافه می‌شوند، بر بازی شخص اثر منفی می‌گذارند که در نتیجه بازی سخت تر خوهد شد.

سبدهای مختص مرحله دوم، دارای حرکت چرخان و دایره‌ای هستند. در صورت برخورد با این نوع سبد، از امتیاز شخص کاسته می‌شود. علاوه بر این، با انجام برخورد، تمام عناصر دیگر بازی از صفحه خارج می‌شوند و تنها ماسک‌های عصبی که از برخورد با سبد به درون

صفحه بازی پرتاب می‌شوند، در صفحه باقی می‌مانند. تعداد این ماسک‌ها پنج عدد است. این ماسک‌ها قصد حمله به خط را دارند. به همین علت، با حرکت خط توسط بازیکن در هر جهت، ماسک‌ها نیز در جهت نزدیک شدن به خط به منظور حمله، حرکت می‌کنند.

فرد بازیکن تا قبل از حمله و برخورد ماسک‌ها به خط، مهلت دارد تا از خود در برابر حمله ماسک‌ها دفاع کند. بازیکن با فشار دادن یک کلید و نگه داشتن آن، خود را جمع می‌کند و به شکل دایره درمی‌آید و حرکتی چرخشی و سریع انجام می‌دهد. همچنین می‌تواند در این حالت خود به سمت ماسک‌ها حرکت کند تا آن‌ها را از صفحه به بیرون پرتاب کند. در این صورت زمان کمتری از دست خواهد داد و می‌تواند برای کسب امتیاز و ادامه بازی زمان بیشتری داشته باشد. با این حال، در صورت عدم حرکت و ثابت ماندن نیز، ماسک‌ها به خط نزدیک می‌شوند و باز به بیرون صفحه پرتاب می‌شوند.

درصورت عدم انجام عمل دفاع از جانب بازیکن، ماسک‌ها به خط نزدیک می‌شوند و با برخورد اولین ماسک به خط، خط به گوشه‌ای پرتاب می‌شود. در طراحی حرکت حاصل از حمله ماسک‌ها سعی بر این بوده که فشار ناشی از ضربه و برخورد ماسک به خط حس شود. به همین دلیل ماسک به هر نقطه از خط که برخورد کند، خط در آن نقطه دچار خمیدگی می‌شود و در جهتی که به آن فشار وارد می‌شود پرتاب می‌شود، تا به بیننده حس ضربه القا شود. در هر صورت، پس از حمله ماسک‌ها یا دفاع خط و خارج شدن ماسک‌ها، بازی به حالت عادی باز می‌گردد و ورود سبدها دوباره از سر گرفته می‌شود. در این مرحله نیز برای حضور سه نوع سبد در صفحه، نکات مرحله قبل در نظر گرفته شده است. در انتها با پایان موسیقی بازی به اتمام می‌رسد و تمام المان‌ها به جز خط از صفحه خارج می‌شود و امتیاز کل شخص نمایش داده می‌شود.

۳-۷- هنر

همان گونه که قبلا نیز گفته شد، هدف اصلی بازی معرفی هنر بومی آفریقا است. شاخص ترین عوامل این بازی نیز موسیقی و صنایع دستی مردم آفریقا است. از مشخصه‌های هنر آفریقا تنوع رنگ است، به همین دلیل در این بازی سعی شده از عناصر گرافیکی که بتواند در عین سادگی، ذهن مخاطب را سریعا به سوی هنر مردم این منطقه هدایت کند، استفاده شود. سبدها و ماسک‌های آفریقایی از مهم‌ترین عوامل استفاده شده در این بازی هستند که طراحی آن‌ها با الهام از هنر آفریقایی صورت گرفته است.

اولین و مهم‌ترین عنصر بازی، خطی است که کنترل آن توسط شخص انجام می‌شود. طراحی آن با الهام از طبیعت و به منظور القا کردن حس حرکت و پویایی انجام گرفته است که نوع حرکت آن القا گر حرکت ماهی است. در زیر، چند نمونه از حرکت این خط دیده می‌شود.

شکل ۷- نمونه هایی از نحوه حرکت خط

در مرحله اول در صورتی که خط با عاملی که باعث کاسته شدن امتیاز می‌شود، برخورد کند، لرزشی در خط ایجاد می‌شود که در زیر دو نمونه از آن آمده است.

شکل ۸- ضربه خوردن خط در مرحله اول در زمان کاهش امتیاز

از دیگر عناصر مرحله اول که در زیرتصویری از آن آورده شده است، سبدی است که برخورد با آن، باعث افزایش امتیاز می‌شود.

شکل ۹- عامل افزایش دهنده امتیاز

در مقابل، در زیر تصویری از عامل کاهش دهنده امتیاز که بازیکن باید از برخورد با آن پرهیز کند، آورده شده است.

شکل ۱۰- نوع اول عامل کاهش دهنده امتیاز

پس از پایان مرحله اول و در زمان شروع مرحله دوم، عامل تشویق کننده که دستی است که تامبورین می‌نوازد، وارد بازی می‌شود .برای مدت زمانی کوتاه، انیمیشنی برای نواختن تامبورین اجرا می‌شود و سپس از صحنه بازی خارج می‌شود. تصویر آن در زیر آورده شده است.

شکل ۱۱- عامل تشویق کننده در زمان تغییر مرحله بازی

در مرحله دوم، نوع دیگری از سبد که بر نحوه بازی تاثیر منفی دارد، وارد صحنه می‌شود که به صورت زیر است:

شکل ۱۲- نوع دوم عامل کاهش دهنده امتیاز

پس از برخورد با سبدهای از نوع بالا، ماسک‌هایی عصبی به بیرون پرتاب می‌شوند که قصد حمله به سمت خط را دارند و به سمت خط حرکت می‌کنند. تصاویر این پنج ماسک در زیر آمده است.

شکل ۱۳- تصاویر عوامل حمله کننده در مرحله دوم

۳-۸- صدا و موسیقی

موسیقی بازی، موسیقی بومی آفریقا است که عموما پرکاشن و درام است و مخاطب را کاملا درگیر بازی می‌کند. علاوه بر این درزمانی‌که مرحله بازی عوض می‌شود، انیمیشنی اجرا می‌شود که تامبورین می‌نوازد.

۳-۹- کنترل‌های بازی

کنترل هایی که توسط شخص بازیکن صورت می‌گیرد، عبارت است از: هدایت خط از طریق حرکت نشانه‌گر و واکنش دفاعی در برابر حمله عناصر منفی (ماسک‌های عصبی) از طریق کلیک کردن و نگه داشتن نشانه‌گر.

۳-۱۰- مخاطب بازی

در طراحی این بازی سعی شده از عناصری استفاده شود که بازی را برای همه گروه‌های سنی امکان‌پذیر سازد.

۳-۱۱- جمع‌بندی و نتیجه‌گیری

در این فصل، در مستند طراحی بازی به شرح مفصل سناریو و داستان بازی دوبعدی طراحی شده پرداخته شد. از جمله موارد مطرح شده در این مستند، شرح داستان بازی، کاراکترها، طراحی مراحل بازی، روند بازی، هنر و گرافیک، موسیقی، کنترل‌های بازی و مخاطب بازی است. در طراحی این بازی سعی بر آن بوده که بتوان قابلیت‌های جدید HTML5 و عنصر canvas به‌خوبی نشان داده شود. در فصل چهارم، به شرح جزئیات پیاده‌سازی بازی با استفاده از زبان جاوا اسکریپت پرداخته خواهد شد.

فصل سوم

پیاده‌سازی بازی

۴-۱- مقدمه

در فصل اول به توابع و امکانات موجود برای استفاده از canvas برای ترسیم شکل های گرافیکی و ایجاد انیمیشین اشاره شد. در فصل دوم نیز بازی مورد نظر، جهت استفاده از این امکانات شرح داده شد. حال زمان آن رسیده است که از رابط کاربری زمینه دو بعدی canvas استفاده کرده و پیاده سازی بازی را انجام دهیم تا از این طریق به طور عملی آشکار شود که این امکانات به چه صورت و تا چه حد کارا هستند.

لازم به ذکر است در پیاده‌سازی این بازی از هیچ موتور بازی استفاده نشده است و تمام برخوردها و انواع حرکات به عنوان جزیی از بازی پیاده‌سازی شده است که البته علت آن نبود چنین ابزاری در محیط canvas است.

۴-۲- پیاده‌سازی خط کنترل شده توسط بازیکن

همانطور که قبلا نیز گفته شد، در پیاده‌سازی عاملی که کنترل آن توسط شخص بازیکن صورت می‌گیرد، به این نکته توجه شده است که حرکت بسیار نرم صورت گیرد تا تداعی کننده حرکت ماهی به عنوان نمادی از طبیعت و حرکت و پویایی باشد و در همین راستا مطالعات بسیاری انجام شد که در نهایت خط به شیوه زیر پیاده‌سازی شد:

برای پیاده‌سازی خط اصلی که حرکت را بتوان با استفاده از آن نشان داد، ازمسیری به شکل زنجیری از نقطه‌ها استفاده شد تا به هم متصل باشند و بتوانند به دنبال هم حرکت کنند. در نتیجه خط انعطاف خواهد داشت.

اما برای پیاده‌سازی حالت ماهی و در واقع برای پیاده‌سازی قطرهای متفاوت در طول خط از مسیر دیگری به نام pathWrap استفاده شد، که دورتا دور مسیر اصلی زنجیره‌ای قرار دارد تا شکل مورد نظر به دست آید.

در ابتدا مسیر اصلی که حرکت خط به وسیله آن انجام می‌شود، ساخته شد. مسیری متشکل از سی و پنج نقطه که اندازه آن متناسب با سایز در نظر گرفته شده برای خط است. سپس برای ایجاد مسیر دور مسیر اصلی نیاز به روشی بود که بتوان دو نقطه یکی در بالا و دیگری در پایین هر نقطه از مسیر اصلی به فاصله مساوی قرار داد تا همراه با نقاط خط اصلی حرکت کنند. فاصله این نقاط تا مسیر اصلی نشانه‌گر، قطر خط در نقطه ی مورد نظر است. برای داشتن بیشترین انعطاف و به وجود نیامدن زوایای تند در زمان حرکت خط اصلی در جهات مختلف ، نقاط مسیر دور ، باید در راستای نقاط اصلی و در واقع هر سه نقطه در یک خط قرار گیرند. در شکل زیر محل قرارگیری نقاط به طور فرضی رسم شده است.

شکل ۱۴- نحوه قرار گیری نقاط بر روی مسیر اصلی و مسیر دور

در زیر نحوه‌ی به‌دست آوردن موقعیت و زاویه نقاط pathWrap آورده شده است. builderVector نام برداری است که برای به دست آوردن این نقاط ایجاد شده است.

برای به دست آوردن زاویه نقاط مسیر دور نسبت به مسیری اصلی به طوری که نقاط بالایی و پایینی هر نقطه روی مسیر اصلی در یک راستا و بر روی یک خط قرار گیرند، ابتدا دو نقطه همسایه نقطه مورد نظر بر روی مسیر اصلی را در نظر می‌گیریم و بردار حاصل از تفاضل این نقاط و نقطه مورد نظر را به دست می‌آوریم. زاویه بین این دو بردار را به دست می‌آوریم. حال برای به دست آمدن زاویه builderVector، زاویه به دست آمده را نصف می‌کنیم و با اندازه زاویه بردار دوم جمع می‌کنیم. حال از آن جایی که این دونقطه در بالا و پایین نقطه اصلی و به فاصله مساوی از آن قرار دارند، به اندازه طول بردار builderVector به نقطه می‌افزاییم و از آن می‌کاهیم تا مختصات دو نقطه روی مسیر مشخص شود. در زیر پیاده‌سازی مربوط به این قسمت آورده شده است.

```
   var size = 35;
   var segments = path.segments;
   var wrapSegments = pathWrap.segments;
   var start = new Point(view.center.x/10,view.center.y);
   for (var i = 0; i < size; i++){
      path.add(start.x + i ,start.y + 20);
      if(i>1){
         var vector1 = new Point(segments[i-2].point.x-segments[i-
1].point.x,segments[i-2].point.y-segments[i-1].point.y);
         var vector2 = new Point(segments[i].point.x-segments[i-
1].point.x,segments[i].point.y-segments[i-1].point.y);
         var angle1 = (vector1.angle<0)?(vector1.angle+360):(vector1.angle);

         var angle2 = (vector2.angle<0)?(vector2.angle+360):(vector2.angle);

         var doubleangle = angle1-angle2;
         if(doubleangle<0){
            doubleangle+=360;
         }
         var buildervector = new Point();
         buildervector.angle = doubleangle/2 + angle2;
         buildervector.length = 10;
         pathWrap.add(new Point(segments[i-
1].point.x+buildervector.x,segments[i-1].point.y+buildervector.y));
         pathWrap.insert(0,new Point(segments[i-1].point.x-
buildervector.x,segments[i-1].point.y-buildervector.y));
      }
   }
pathWrap.closed = true;
```

در هر فریم، در صورتی که خط توسط شخص بازیکن جابه‌جا شده باشد، باید این جابه‌جایی به صورت زنجیروار، در تمام نقاط مربوط به مسیر اصلی و مسیر دور نیز صورت گیرد. به منظور داشتن حداکثر انعطاف و عدم ایجاد زوایای تند، جابه‌جایی نقاط به صورت زیر پیاده‌سازی شد:

از رخداد جابه‌جایی نشانه‌گر، برای آگاهی از جابه جا شدن نشانه گر استفاده می‌شود. سپس مختصات نقطه اول خط اصلی با این مختصات جایگزین می‌شود. حال باید مختصات سایر نقاط نیز با توجه به نحوه جابه جایی نقطه اول، تغییر کند.

بدین منظور، بردار تفاضل نقطه جدید سرخط و نقطه بعدی آن را به دست آورده، عدد ده برای طول این فاصله در نظر گرفته شده است، در صورتی که مقدار جابه جایی بیشتر از این عدد باشد طول را همان ده در نظر گرفته در غیر این صورت طول بردار تفاضل همان مقدار واقعی خواهد بود. حال موقعیت جدید نقطه دوم برابر تفاضل موقعیت جدید نقطه اول و بردار تفاضل خواهد بود. زاویه بین این دو نقطه نیز همان زاویه بردار تفاضل خواهد بود. با همین روش، موقعیت جدید تمام نقاط روی مسیر اصلی به دست می‌آید. حال با داشتن مختصات تمام نقاط مسیر اصلی به همان شیوه‌ای که در بالا گفته شد، مختصات نقاط روی مسیر دور به روز می‌شود. این عمل در هر فریم در تابع به روز رسانی pathUpdate صورت می‌گیرد که پیاده‌سازی آن در زیر آورده شده است.

```
        for (var j = 0; j < size - 1; j++) {
            var nextSegment = segments[j + 1];
            var position = path.segments[j].point;
            var angleP = new Point(position.x - nextSegment.point.x,position.y -
nextSegment.point.y);
            var vector = new Point();
            vector.angle= angleP.angle;
            var length=10;
            if(mode=='endofGameJam'){
               if(endofGameJamLength>0){
```

```
                endofGameJamLength -= 0.005;
            }
            length = endofGameJamLength;
        }
        vector.length= angleP.length<length?angleP.length:length;
        nextSegment.point = new Point(position.x - vector.x,position.y -
vector.y);

    }
```

۴-۳- نحوه ورود سبدها به صفحه بازی

در انتخاب قاعده‌ای که برای محاسبه نحوه ورود سبدها به صفحه در نظر گرفته شده است، سعی بر آن بوده که نوعی تعادل در تعداد سبدهای مثبت و منفی وجود داشته باشد. بدین منظور روش زیر برای پیاده‌سازی به کار رفته است:

ناحیه‌ای در خارج صفحه به عرض صد پیکسل در نظر گرفته شده است. شماره‌گری به منظور محاسبه فاصله‌های زمانی که سبد تولید می‌شود در نظر گرفته شده است. به ازای هر دویست فریم یک سبد تولید می‌شود. برای اینکه تعادل در ورود انواع مختلف سبد موجود باشد، احتمالی برای تعیین نوع سبد در نظر گرفته شده است، در ابتدا این احتمال برابر مقدار یکسان و برابر ۰٫۵ است . با مشخص شدن نوع اولین سبد وارد شده به صفحه، احتمال ورود گونه مورد نظر به ۲۷ درصد کاهش می‌یابد و در مقابل احتمال ورود نوع دیگر سبد به ۰٫۷۳ افزایش می‌یابد. اما اگر باز هم با وجود کاهش احتمال ورود، از همان سبدهای نوع قبلی وارد صفحه شود، این بار احتمال صفر خواهد شد. در نتیجه هیچ گاه بیشتر از دو عدد از یک نوع سبد پشت سر هم وارد صفحه نخواهد شد. پس از آمدن سبد نوع دیگر، دوباره احتمال به ۲۷ و ۷۳ بدل خواهد شد و این روند به همین شیوه ادامه خواهد یافت.

در مرحله دوم، به دلیل اضافه شدن نوع دیگری از سبدها، از آن جایی که این سبدها نیز تاثیری منفی بر بازی دارند، همان احتمال برای این نوع سبد نیز به کار می‌رود. با این

تفاوت که در صورت انتخاب نشدن سبد با اثر مثبت با احتمال ثابت ۲۷و ۷۳ درصد، بین این دو نوع سبد تصمیم گیری خواهد شد. در زیر پیاده‌سازی این قسمت آورده شده است.

```
      if(level ==1){

if(++basketMakingFrameCounter==Math.floor(1000/basketArrivalSpeed)){
            basketMakingFrameCounter = 0;
            var newPos = random();
            for (var h = 0; h < baskets.length; h++) {
               while((Math.abs(newPos.x-baskets[h].element.position.x)<85)
&& (Math.abs(newPos.y-baskets[h].element.position.y)<85)){
                  newPos=random();
                  h=0;
               }
            }
            var rand = Math.random();
            if(rand>goodOrBadProbability){
                        if(goodOrBadProbability<0.5){
               goodOrBadProbability = 0.5 + 0.23;
               }else{
                  goodOrBadProbability += 0.23;
               }
               var newPlacedSymbol;
               newPlacedSymbol = greenSymbol.place(newPos);
               newPlacedSymbol.scale(Math.random() * (.7-.5) + .5);
               baskets.push(new GreenBasket(newPlacedSymbol));
            }else{
               if(goodOrBadProbability>0.5){
                  goodOrBadProbability = 0.5 - 0.23;
               }else{
                  goodOrBadProbability -= 0.23;
               }
```

```
                var newPlacedSymbol1;
                newPlacedSymbol1 = brownSymbol.place(newPos);
                newPlacedSymbol1.scale(Math.random() * (.7-.5) + .5);
                baskets.push(new BrownBasket1(newPlacedSymbol1));
              }
            }
        if(level==2){
            if(masks.length==0 && mode!="attackRecovery" &&
mode!="attacked"){

    if(++basketMakingFrameCounter==Math.floor(1000/basketArrivalSpeed)){
                basketMakingFrameCounter = 0;
                var newPos1 = random();
                for (var g = 0; g < baskets.length; g++) {
                   while((Math.abs(newPos1.x-baskets[g].element.position.x)<85)
&& (Math.abs(newPos1.y-baskets[g].element.position.y)<85)){
                      newPos1=random();
                      g=0;
                   }
                }
                var rand1 = Math.random();
                if(rand1>goodOrBadProbability){
                   if(goodOrBadProbability<0.5){
                      goodOrBadProbability = 0.5 + 0.23;
                   }else{
                      goodOrBadProbability += 0.23;
                   }
                   var newPlacedSymbol2;
                   newPlacedSymbol2 = greenSymbol.place(newPos1);
                   newPlacedSymbol2.scale(Math.random() * (.7-.5) + .5);
                   baskets.push(new GreenBasket(newPlacedSymbol2));
                }else{
```

```
            if(goodOrBadProbability>0.5){
               goodOrBadProbability = 0.5 - 0.23;
            }else{
               goodOrBadProbability -= 0.23;
            }
            var rand2 = Math.random();
            if(rand2>badOrMasksProbability){
               var newPlacedSymbol3;
               newPlacedSymbol3 = brownSymbol.place(newPos1);
               newPlacedSymbol3.scale(Math.random() * (.7-.5) + .5);
               baskets.push(new BrownBasket1(newPlacedSymbol3));
            }else{
               var newPlacedSymbol4;
               newPlacedSymbol4 = germanySymbol.place(newPos1);
               newPlacedSymbol4.scale(Math.random() * (.75-.6) + .6);
               baskets.push(new BrownBasket2(newPlacedSymbol4))
            }
         }
      }
```

۴-۴- پیاده‌سازی حرکت سبدها

در مرحله اول دو نوع سبد وجود دارد. نوعی از این سبدها دارای حرکت چرخشی هستند. نوع دیگر سبدها دارای حرکتی هستند که حس معلق بودن را القا می‌کند. علاوه بر حرکت مخصوص هر نوع سبد، برای اینکه حس حرکت صفحه به بیننده منتقل شود، سبدها از سمت راست وارد صفحه می‌شوند و از سمت چپ از صفحه خارج می‌شوند. به منظور سرعت بخشیدن به این عمل، در هر فریم، سبدها هشت پیکسل در راستای افقی جابه‌جا می‌شوند. برای اینکه این حرکات به صورت انیمیشن در مدت زمان حضور سبد در صفحه انجام گیرد، این اعمال در هر فریم تکرار می‌شوند. سپس با خارج شدن هر سبد از صفحه، آن سبد از

آرایه شامل سبدها حذف می‌شود. نحوه حرکت سبدهای افزایش دهنده امتیاز، به شکل زیر پیاده‌سازی شده است.

```
GreenBasket.prototype.update = function(){
    this.element.rotate(20);
    this.element.position.x -=8;
    if (this.element.bounds.right < 0) {
        this.element.visible = false;
        var n = baskets.indexOf(this);
        baskets.splice(n,1);
        this.element.remove();
```

در مرحله دوم بازی نوع دیگری از سبدها به بازی اضافه می‌شوند که حرکتی مشابه سبدهای مرحله اول دارند و به همان صورت بالا پیاده‌سازی می‌شود.

اما پیاده‌سازی نحوه حرکت سبدهای کاهش دهنده امتیاز که به نوعی حس معلق بودن را القا می‌کنند، به صورت زیر است:

```
    if(this.movement == 1){
        this.element.position.x -=.5;
        this.element.position.y +=1;
        this.moveCount++;
        if(this.moveCount==10){
            this.movement=2;
            this.moveCount=0;
        }
    }
    if(this.movement ==2){
        this.element.position.x +=.5;
        this.element.position.y -=1;
        this.moveCount++;
        if(this.moveCount==10){
```

```
            this.movement=3;
            this.moveCount=0;
        }
    }
    if(this.movement==3){
        this.element.position.x +=1;
        this.element.position.y +=.5;
        this.moveCount++;
        if(this.moveCount==10){
            this.movement=4;
            this.moveCount=0;
        }
    }
    if(this.movement==4){
        this.element.position.x -=1;
        this.element.position.y -=.5;
        this.moveCount++;
        if(this.moveCount==10){
            this.movement=1;
            this.moveCount=0;
        }
    }
    this.element.position.x -=12 ;

    if (this.element.bounds.right < 0) {
        this.element.visible = false;
        var n = baskets.indexOf(this);
        baskets.splice(n,1);
        this.element.remove();

    }
```

۴-۵- مدیریت برخورد بین خط و سبد

در مرحله اول بازی، نکته مهم دیگر مدیریت برخورد بین خط و سبدها است، چرا که برخورد بین این دو عنصر، مشخص کننده روش امتیاز دهی است.

برخورد به نحوی تعریف شده است که در صورتی که نقطه آغازین خط در فاصله‌ای به شعاع ۳۰ پیکسل از خط دور هریک از سبدها قرار گیرد، برخورد تلقی می‌شود. در صورتی که برخورد صورت گیرد، سبد مورد نظر، از آرایه‌ای که شامل تمام سبدهای موجود در صفحه است حذف می‌شود. در صورت برخورد با سبدهای کاهنده امتیاز، نوعی لرزش و حرکت موجی درخط ایجاد می‌شود که نمایانگر ضربه خوردن خط است. علاوه بر این با برخورد با هر نوع سبد، قاعده‌ای که برای امتیازدهی در نظر گرفته شده است، اعمال می‌شود.

ضربه خوردن خط در مرحله اول با تصادفی کردن طول بردار builderVector که در زمان رسم خط درباره آن صحبت شد، ایجاد می‌شود. در حالت عادی طول این بردار در هر نقطه مشخص است. این عمل حس لرزش و موج را القا می‌کند.

در زیر نحوه پیاده‌سازی آورده شده است:

```
if(brown1hit){
   buildervector1.length += -3/2+3*Math.random();
   buildervector2.length += -3/2+3*Math.random();
   if((new Date().getTime() - brown1hitTime) > 1000){
      brown1hit = false;
   }
}
```

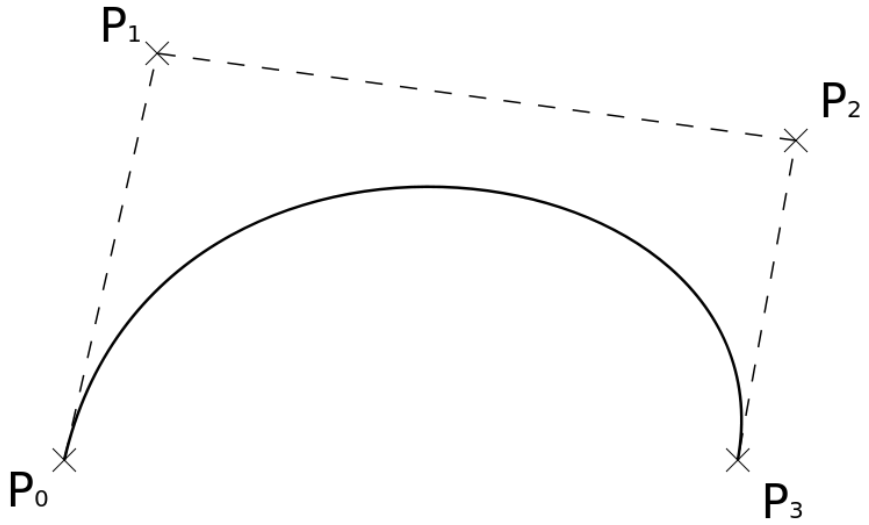
۴-۶- نحوه حرکت ماسک‌ها

در مرحله دوم نوع جدیدی از سبدها وارد بازی می‌شوند. پس از برخورد با این نوع سبد، پنج ماسک در موقعیت برخورد ایجاد می‌شوند. از آن‌جایی که این ماسک‌ها قصد حمله به خط را دارند، بر روی منحنی‌های bezier درجه سوم به خط نزدیک می‌شوند تا به آن برخورد کنند. با حرکت و جابه‌جایی خط در هر لحظه، منحنی bezier جدیدی تا موقعیت جدید خط رسم می‌شود و ماسک بر روی این منحنی جدید حرکت می‌کند و در هر لحظه با رسم منحنی‌های جدید، سرعت حرکت ماسک‌ها نیز با ضریبی ثابت، افزایش می‌یابد.

در زیرمنحنی bezier و نحوه پیاده‌سازی این منحنی توضیح داده شده است.

۴-۷- منحنی bezier

منحنی bezier یک منحنی پارامتری است که عموما در گرافیک کامپیوتری و حوزه‌های مربوطه کاربرد دارد. در گرافیک برداری، منحنیbezier به منظور مدل کردن منحنی‌های نرم که به‌طور نامحدود مقیاس پذیرند، به کار می‌رود. مسیرها ترکیبی از منحنی‌های bezier متصل به هم هستند. همچنین از این منحنی‌ها در انیمیشن به عنوان ابزاری در کنترل حرکت استفاده می‌شود.



شکل ۱۵- منحنی bezier درجه سوم

۴-۸- کاربرد منحنی bezier در ایجاد انیمیشن

در برنامه‌های انیمیشنی، مانند فلش، منحنی‌های bezier برای مشخص کردن حرکت به کار می‌روند. کاربر مسیر موردنظر را به شکل منحنی bezier مشخص می‌کند و برنامه، فریم‌های لازم برای حرکت شی در راستای مسیر را ایجاد می‌کند.

۴-۸-۱- تعریف ریاضی منحنی bezier

منحنی bezier با استفاده از مجموعه‌ای از نقاط کنترلی P_0 تا P_n تعریف می‌شود (n مشخص کننده درجه منحنی است). اولین و آخرین نقطه کنترلی همواره نقاط انتهایی منحنی هستند. اما نقاط کنترلی میانی بر روی منحنی قرار نمی‌گیرند.

با داشتن دو نقطه کنترلی P_0 و P_1 ، منحنی bezier، همان خط مستقیمی است که این دو نقطه را به هم وصل می‌کند.

این منحنی به صورت زیر تعریف می‌شود:

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , \ t \in [0,1]$$

که معادل همان درون یابی خطی است.

این منحنی مسیر طی شده توسط تابع B(t) است با سه نقطه کنترلی P۰ ، P۱ و P۲.

$$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2] \ , \ t \in [0,1]$$

درواقع می‌توان آن را به صورت درون‌یاب خطی نقاط متناظر بر روی منحنی bezier خطی گذرنده از دو نقطه P۰، P۱ و دو نقطه P۲، P۱ در نظر گرفت. با مرتب کردن دوباره منحنی فوق به معادله زیر می‌رسیم.

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \ , \ t \in [0,1].$$

با مشتق گرفتن از تابع بالا نسبت به t داریم:

$$\mathbf{B}'(t) = 2(1-t)(\mathbf{P}_1 - \mathbf{P}_0) + 2t(\mathbf{P}_2 - \mathbf{P}_1)\,.$$

که از این معادله می‌توان نتیجه گرفت که مماس بر منحنی گذرنده از P۰ و P۲ در نقطه P۱ دارای شیب صفر است.

با افزایش t از صفر تا یک ، منحنی از P۰ در جهت P۱ شروع به حرکت می‌کند. سپس دچار خمیدگی می‌شود تا به نقطه P۲ برسد.

چهار نقطه P۰ ، P۱، P۲ وP۳ مشخص کننده منحنی bezier درجه سوم هستند. منحنی از نقطه P۰ به سمت P۱ حرکت را آغاز می‌کند و به نقطه P۳ در جهت آمده از نقطه P۲ می‌رسد. منحنی از دو نقطه P۲ و P۱ عبور نخواهد کرد، چرا که این دو نقطه، فقط اطلاعات مربوط به جهت حرکت را فراهم می‌کنند. فاصله بین P۱ و P۰ مشخص کننده مقدار طی شده در جهت P۲ قبل از حرکت به سمت P۳ است.

منحنی درجه سوم می‌تواند به عنوان ترکیبی خطی از دو منحنی درجه دوم به صورت زیر تعریف شود:

$$\mathbf{B}(t) = (1-t)\mathbf{B}_{\mathbf{P_0},\mathbf{P_1},\mathbf{P_2}}(t) + t\mathbf{B}_{\mathbf{P_1},\mathbf{P_2},\mathbf{P_3}}(t) \ , \ t \in [0,1].$$

تعریف دقیق تابع به صورت زیر است:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , \ t \in [0,1].$$

به منظور پیاده‌سازی حرکت بر روی منحنی bezier درجه سوم از روش زیر استفاده شده است:

چهار نقطه P_0 ، P_1، P_2 و P_3 درنظر گرفته شده است.

P_0 به عنوان نقطه آغازی با مختصات X_0 و Y_0 در نظر گرفته شده است.

P_3 به عنوان نقطه پایانی با مختصات X_3 و Y_3 در نظر گرفته شده است.

P_1 و P_2 نقاط کنترلی هستند که منحنی از آن‌ها عبور نمی کند و تنها برای مشخص کردن میزان انحنای منحنی به کار می‌روند.

بعد از مشخص شدن مختصات این چهار نقطه، نیاز به محاسبه شش ضریب داریم تا بتوانیم مختصات نقاطی که بر روی مسیر قرار دارند را به دست بیاوریم و مسیر حرکت مشخص شود. این ضرایب از طریق فرمول‌های زیر محاسبه می‌شوند.

```
var cx = 3 * (p1.x - p0.x)
var bx = 3 * (p2.x - p1.x) - cx;
var ax = p3.x - p0.x - cx - bx;


var cy = 3 * (p1.y - p0.y);
var by = 3 * (p2.y - p1.y) - cy;
var ay = p3.y - p0.y - cy - by;
```

حال با داشتن این شش ضریب می‌توان مقادیر x(t) وy(t) را با تغییردادن مقدار t در معادله زیر به دست آورد.

t نشان دهنده حرکت در طول زمان است:

$$x(t) = ax*t^3 + bxt^2 + cx*t + x_0$$

$$y(t) = ay*t^3 + byt^2 + cy*t + y_0$$

t با توجه به مقدار مورد نظر برای سرعت حرکت شی، افزایش می‌یابد.

برای نمونه چهار نقطه زیر را در نظر می‌گیریم:

```
var p0 = {x:60, y:10};
var p1 = {x:70, y:200};
var p2 = {x:125, y:295};
var p3 = {x:350, y:350};
```

شی ball را با درنظر گرفتن مختصات اولیه x و y و سرعت اولیه ۰،۱ ایجاد می‌کنیم که نشان می‌دهد، شی از صد نقطه عبور می‌کند تا به P_3 برسد. همچنین مشخص است که حرکت از نقطه P_0 آغاز می‌شود.

```
var ball = {x:0, y:0, speed:.01, t:0};
```

سپس ضرایب منحنی bezier را با استفاده از فرمول‌هایی که قبلا معرفی شد، محاسبه می‌کنیم.

سپس مقدار t شی ball را به طور محلی ذخیره می‌کنیم و با استفاده از آن مختصات x(t) و y(t) نقاط مسیر را به دست می‌آوریم.

```
var t = ball.t;
```

سپس مقدار سرعت را با مقدار t جمع می‌کنیم تا نقطه بعدی روی مسیر به دست آید.

```
ball.t += ball.speed;
var xt = ax*(t*t*t) + bx*(t*t) + cx*t + p0.x;
var yt = ay*(t*t*t) + by*(t*t) + cy*t + p0.y;
```

با استفاده از دو فرمول بالا مختصات نقاط به دست می‌آید.

و در انتها شی ball را بر روی canvas رسم می‌کنیم:

```
context.arc(xt,yt,5,0,Math.PI*2,true);
```

در انتها، پیاده‌سازی کلی منحنی به صورت زیر خواهد بود:

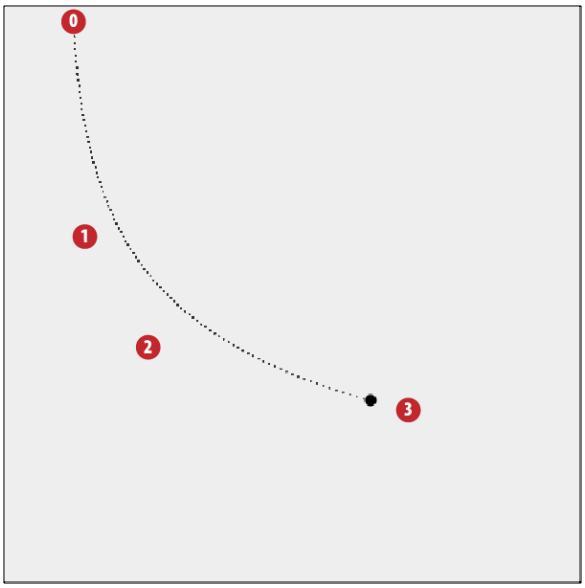
```
<!doctype HTML>
<HTML lang="en">
<head>
<meta charset="UTF-8">
<title>CH5EX11: Moving On A Cubic Bezier Curve </title>
<script src="/book/2471/OEBPS///modernizr-1.6.min.js"></script>
<script type="text/javascript">
window.addEventListener('load', eventWindowLoaded, false);
function eventWindowLoaded() {
  canvasApp();

}
function canvasSupport () {
    return Modernizr.canvas;
}
```

```
function canvasApp() {
  if (!canvasSupport()) {
        return;
       }
  var pointImage = new Image();
  pointImage.src = "point.png";

  function  drawScreen () {
     context.fillStyle = '#EEEEEE';
     context.fillRect(0, 0, theCanvas.width, theCanvas.height);
     //Box
     context.strokeStyle = '#000000';
     context.strokeRect(1,  1, theCanvas.width-2, theCanvas.height-2);

     var t = ball.t;

     var cx = 3 * (p1.x - p0.x)
     var bx = 3 * (p2.x - p1.x) - cx;
     var ax = p3.x - p0.x - cx - bx;

     var cy = 3 * (p1.y - p0.y);
     var by = 3 * (p2.y - p1.y) - cy;
     var ay = p3.y - p0.y - cy - by;

     var xt = ax*(t*t*t) + bx*(t*t) + cx*t + p0.x;
     var yt = ay*(t*t*t) + by*(t*t) + cy*t + p0.y;

     ball.t += ball.speed;

     if (ball.t > 1) {
       ball.t = 1;
     }
```

```
//draw the points
context.font ="10px sans";
context.fillStyle = "#FF0000";
context.beginPath();
context.arc(p0.x,p0.y,8,0,Math.PI*2,true);
context.closePath();
context.fill();
context.fillStyle = "#FFFFFF";
context.fillText("0",p0.x-2,p0.y+2);

context.fillStyle = "#FF0000";
context.beginPath();
context.arc(p1.x,p1.y,8,0,Math.PI*2,true);
context.closePath();
context.fill();
context.fillStyle = "#FFFFFF";
context.fillText("1",p1.x-2,p1.y+2);

context.fillStyle = "#FF0000";
context.beginPath();
context.arc(p2.x,p2.y,8,0,Math.PI*2,true);
context.closePath();
context.fill();
context.fillStyle = "#FFFFFF";
context.fillText("2",p2.x-2, p2.y+2);

context.fillStyle = "#FF0000";
context.beginPath();
context.arc(p3.x,p3.y,8,0,Math.PI*2,true);
context.closePath();
context.fill();
```

```
    context.fillStyle = "#FFFFFF";
    context.fillText("3",p3.x-2, p3.y+2);

    //Draw points to illustrate path
    points.push({x:xt,y:yt});
    for (var i = 0; i< points.length; i++) {
      context.drawImage(pointImage, points[i].x, points[i].y,1,1);
    }
    context.closePath();

    //Draw circle moving
    context.fillStyle = "#000000";
    context.beginPath();
    context.arc(xt,yt,5,0,Math.PI*2,true);
    context.closePath();
    context.fill();
  }

  var p0 = {x:60, y:10};
  var p1 = {x:70, y:200};
  var p2 = {x:125, y:295};
  var p3 = {x:350, y:350};
  var ball = {x:0, y:0, speed:.01, t:0};
  var points = new Array();

  theCanvas = document.getElementById("canvasOne");
  context = theCanvas.getContext("2d");

  setInterval(drawScreen, 33);

}
```

```
</script>

</head>
<body>
<div style="position: absolute; top: 50px; left: 50px;">

<canvas id="canvasOne" width="500" height="500">
 Your browser does not support HTML5 Canvas.
</canvas>
</div>
</body>
</HTML>
```

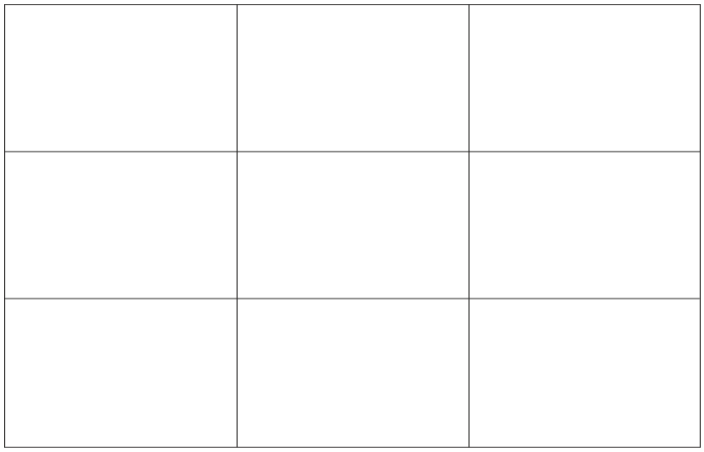


شکل ۱۶- پیاده‌سازی منحنی bezier درجه سوم

در بازی پیاده‌سازی شده، از منحنی bezier درجه سوم برای رسم مسیر حرکت ماسک‌ها در صفحه استفاده شده است. بدین صورت که در لحظه برخورد، نقطه برخورد به عنوان نقطه اول منحنی، موقعیت خط که با استفاده از موقعیت نشانه‌گر مشخص است، به عنوان نقطه

انتهایی منحنی در نظر گرفته می‌شود. برای مشخص کردن موقعیت نقاط کنترلی از روش زیر استفاده شده است:

همانطور که در شکل زیر نیز مشخص شده صفحه canvas به ۹ قسمت تقسیم شده است، برخورد در هر کدام از این نواحی صورت گیرد، به ازای هر یک از ماسک‌ها، یک مسیر حرکت به شکل منحنی قرار دارد. پس در هر ناحیه، پنج منحنی باید رسم شود. برای هر منحنی علاوه برنقاط ابتدایی و انتهایی، دو نقطه کنترلی لازم است. پس برای کل canvas، ۹۰ نقطه کنترلی تعریف شده است.



شکل ۱۷- تقسیم بندی canvas به نه ناحیه

از آن‌جایی که خطوط قصد حمله به خط را دارند، در صورت جابه‌جا شدن خط، منحنی bezier جدیدی از محل فعلی ماسک به موقعیت جدید خط رسم می‌شود. با هر رسم جدید منحنی، سرعت حرکت ماسک نیز، در جهت نزدیک شدن به خط افزایش می‌یابد. این روند تا زمانی که بین خط و ماسک برخورد صورت گیرد ادامه می‌یابد.

در صورتی که خط از خود در مقابل حمله ماسک‌ها دفاع کند، ماسک‌ها با سرعتی تصادفی در یک محدوده مشخص، به بیرون پرتاب می‌شوند. حرکت ماسک‌ها به خارج از صفحه، به صورت زیر است:

در طول مدت زمان حرکت ماسک در canvas، همواره موقعیت فعلی و قبلی ماسک نگه‌داری می‌شود. در زمان برخورد ماسک با خط، برداری بین اولین نقطه برخورد و موقعیت

قبلی ماسک رسم می‌شود و ماسک در جهت بردار رسم شده به بیرون از صفحه پرتاب می‌شود.

پیاده‌سازی نحوه حرکت ماسک‌ها قبل از برخورد باخط و پس از انجام عمل دفاع توسط خط به صورت زیر انجام شده است:

```
function AngryMask(element,p0,p1,p2,p3,speed){
    this.element=element;
    this.type = "mask";
    this.p0 = p0;
    this.p1 = p1;
    this.p2 = p2;
    this.p3 = p3;
    this.p3 = segments[Math.floor(size/2)].point;
    this.cx = 3*(this.p1.x - this.p0.x);
    this.bx = 3*(this.p2.x - this.p1.x) - this.cx;
    this.ax = this.p3.x - this.p0.x - this.cx - this.bx;
    this.cy = 3*(this.p1.y - this.p0.y);
    this.by = 3*(this.p2.y - this.p1.y) - this.cy;
    this.ay = this.p3.y - this.p0.y - this.cy - this.by;
    this.t = 0;
    this.firstTry = true;
    this.speed = speed;
    this.vanishVector=null;
    this.previousPosition = new Point();
    this.dispose = false;
    var randX = 10+(Math.random()*10);
    var randY = 10+(Math.random()*10);
    var rand1X = Math.floor(Math.random()*2);
    var rand1Y = Math.floor(Math.random()*2);
    if(rand1X ==0){
       this.velocityX=randX;
```

```
      }
      else {
         this.velocityX=-randX;
      }
      if(rand1Y==1){
         this.velocityY=randY;
      }else{
         this.velocityY=-randY;
      }
   }
   AngryMask.prototype.update = function(){
      if(mode=="attacked"||mode=="attackRecovery"){
         this.dispose = true;
      }
      if(this.dispose){
         var disposeWhenAttackedVector = new Point(this.element.position.x-
segments[0].point.x,this.element.position.y-segments[0].point.y);
         var normalDisposeWhenAttackedVector =
disposeWhenAttackedVector.normalize();
         normalDisposeWhenAttackedVector.length = 10;
         this.element.position.x += normalDisposeWhenAttackedVector.x;
         this.element.position.y += normalDisposeWhenAttackedVector.y;

if(this.element.bounds.right<0||this.element.bounds.left>x||this.element.bounds.
top>y||this.element.bounds.bottom<0){
            var index = masks.indexOf(this);
            masks.splice(index,1);
            this.element.remove();
         }
      }else{
         if(!this.vanishVector){
```

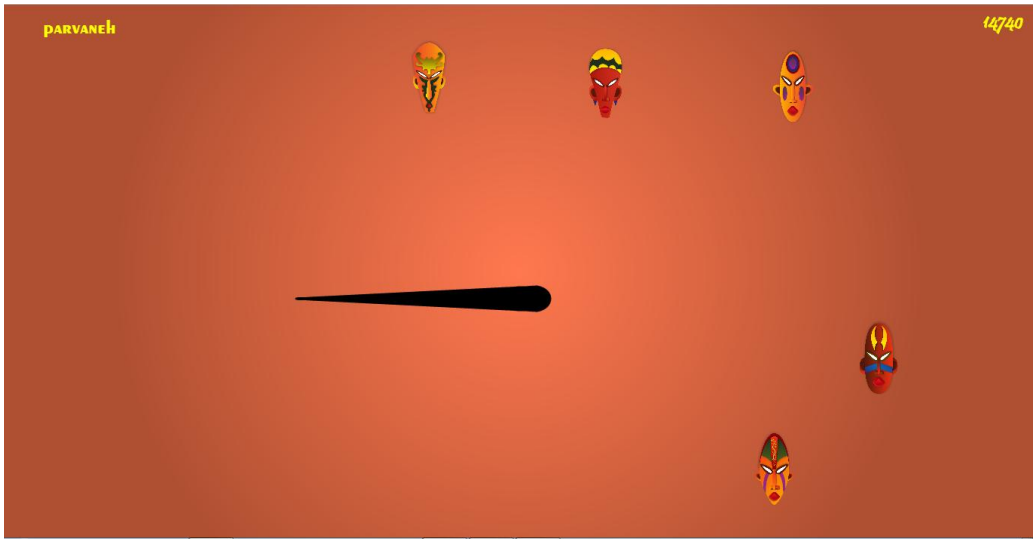
```
            if(this.t<1){
                this.previousPosition.x = this.element.position.x;
                this.previousPosition.y = this.element.position.y;
                this.element.position.x = this.ax*this.t*this.t*this.t +
this.bx*this.t*this.t+this.cx*this.t + this.p0.x;
                this.element.position.y = this.ay*this.t*this.t*this.t +
this.by*this.t*this.t+this.cy*this.t + this.p0.y;
                this.t+=this.speed;
            }else{
                this.firstTry = false;
                this.p0 = this.element.position;
                this.p3 = segments[Math.floor(size/2)].point;
                this.cx = 3*(this.p1.x - this.p0.x);
                this.bx = 3*(this.p2.x - this.p1.x) - this.cx;
                this.ax = this.p3.x - this.p0.x - this.cx - this.bx;
                this.cy = 3*(this.p1.y - this.p0.y);
                this.by = 3*(this.p2.y - this.p1.y) - this.cy;
                this.ay = this.p3.y - this.p0.y - this.cy - this.by;
                this.t = 0;
                this.speed += 0.004;
            }
        }else{
            this.element.position.x+=this.vanishVector.x;
            this.element.position.y+=this.vanishVector.y;

if(this.element.position.x<0||this.element.position.y<0||this.element.position.x>
x||this.element.position.y>y){
                var index1 = masks.indexOf(this);
                masks.splice(index1,1);
                this.element.remove();
            }
        }
```

```
        }
    }
    var attackVector = new Point();
    var attackedEvolutionShape = new Path();
    var attackedEvolutionVectorsToShape = new Array();
    var attackedEvolved = false;
    AngryMask.prototype.hit = function(){
        if(mode == "attacked"){
            return false;
        }
        if(this.t<0.3 && this.firstTry){
            return false;
        }
        var hitResult;
        hitResult                                                    =
pathWrap.hitTest(this.element.position,angryMaskHitOptions);
        if(hitResult){
            if(this.vanishVector){
                return false;
            }
            if(mode=="defence"){
                var vector=new Point();
                vector.x= this.element.position.x-segments[0].point.x;
                vector.y= this.element.position.y-segments[0].point.y;
                this.vanishVector= vector;
                return false;
            }else{
                attackVector.x = this.element.position.x - this.previousPosition.x;
                attackVector.y = this.element.position.y - this.previousPosition.y;
                attackVector.length = 12;
                var index = masks.indexOf(this);
                masks.splice(index,1);
```

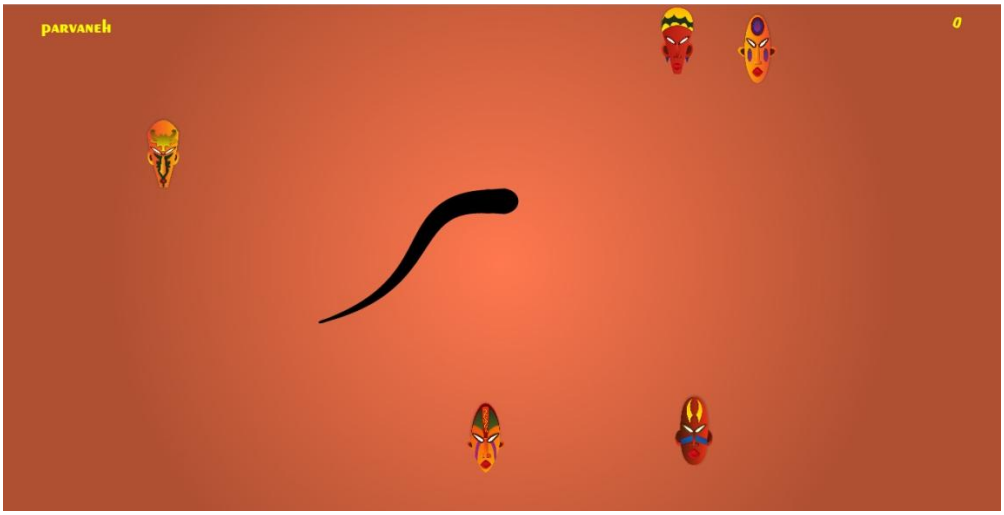
```
            this.element.remove();
            attackedEvolutionShape = new Path();

            var attackVector2 = new Point(attackVector.x,attackVector.y);
            var pathHitIndex = (hitResult.segment.index<size-
2)?(hitResult.segment.index+1):(2*(size-2)-hitResult.segment.index);
            for(var i=pathHitIndex;i<size;i++){
               var temp = (i-pathHitIndex)*(i-pathHitIndex);
               attackVector2.length = Math.abs(100-temp)/3.5;
               if(temp<100){
                  attackedEvolutionShape.add(new  Point(segments[i].point.x  +
attackVector2.x,segments[i].point.y + attackVector2.y));
               }else{
                  attackedEvolutionShape.add(new  Point(segments[i].point.x  -
attackVector2.x,segments[i].point.y - attackVector2.y));
               }
            }
            for(var j=pathHitIndex-1;j>=0;j--){
               var temp2 = (pathHitIndex-j)*(pathHitIndex-j);
               attackVector2.length = Math.abs(100-temp2)/3.5;
               if(temp2<100){
                  attackedEvolutionShape.insert(0,new  Point(segments[j].point.x
+ attackVector2.x,segments[j].point.y + attackVector2.y));
               }else{
                  attackedEvolutionShape.insert(0,new  Point(segments[j].point.x
- attackVector2.x,segments[j].point.y - attackVector2.y));
               }
            }
            attackedEvolutionVectorsToShape = new Array();
            for(var k=0;k<size;k++){
               attackedEvolutionVectorsToShape.push(new
Point(attackedEvolutionShape.segments[k].point.x-
```

```
segments[k].point.x,attackedEvolutionShape.segments[k].point.y-
segments[k].point.y));
                attackedEvolutionVectorsToShape[k].length /=5;
            }
            mode = "attacked";
            attackedEvolved = false;
            return true;

        }
    }
    return false;
  }
```

در زیر دو تصویر از بازی که حرکت ماسک‌ها را نشان می‌دهد، آورده شده است:



شکل ۱۸- تصویر اول از حرکت ماسک‌ها



شکل ۱۹- تصویر دوم از حرکت ماسک‌ها

۴-۹- دفاع خط در مقابله با حمله ماسک ها

پس از برخورد خط با سبدهای جدید مرحله دوم، پنج ماسک در داخل صفحه ظاهر می‌شوند که در جهت نزدیک شدن به خط به حرکت در می‌آیند. در این حالت بازیکن زمان دارد که تا قبل از برخورد اولین ماسک به خط، از خود دفاع کند و ماسک‌ها را به بیرون براند. دفاع به شکل زیر انجام می‌شود:

فرد بازیکن با کلیک کردن و نگه داشتن نشانه‌گر، به حالت دفاع وارد می‌شود، خط با انجام انیمیشنی خود را جمع می‌کند، به شکل دایره درمی‌آید و با سرعت شروع به چرخیدن می‌کند. در این حالت، خط توانایی حرکت به سمت ماسک‌ها و تسریع عمل دفاع را دارد. زیرا با انجام این عمل، ممکن است بتوان ماسک‌ها را زودتر از صفحه به بیرون براند و بازیکن برای ادامه بازی زمان بیشتری داشته باشد. اما در صورت عدم حرکت نیز، دفاع انجام می‌شود. بدین صورت که با نزدیک شدن ماسک‌ها به قصد حمله و برخورد با خط، ماسک‌ها به بیرون پرتاب می‌شوند. نحوه پرتاب شدن ماسک‌ها به بیرون و پیاده‌سازی مربوط به آن، در قسمت قبلی شرح داده شد.

در زیر پیاده‌سازی دفاع خط در مقابل حمله ماسک‌ها آورده شده است:

در هنگام کلیک کردن کاربر، تابع کنترلی زیر خط را به حالت دفاعی برده و یک بردار تبدیل از محل فعلی خط به شکل دفاعی که در کد شکل آن مشخص شده مقدار دهی می‌شود. همچنین متغیر حالت خط به defence تغییر می‌یابد تا در هنگام به‌روز رسانی خط، خط به شکل دفاعی در بیاید.

```
tool.onMouseDown = function(event) {
    if(masks.length>0)
    {
      defenceEvolutionVector.x= eventt.point.x-pointArray[0].x;
      defenceEvolutionVector.y= eventt.point.y-pointArray[0].y;
      defenceEvolved = false;
      mode = "defence";
    }
  }
```

در ضمن اگر کاربر دست از کلیک نشانه گر بردارد، متغیر حالت به normal باز می‌گردد:

```
tool.onMouseUp = function(event){
    mode="normal";
  }
```

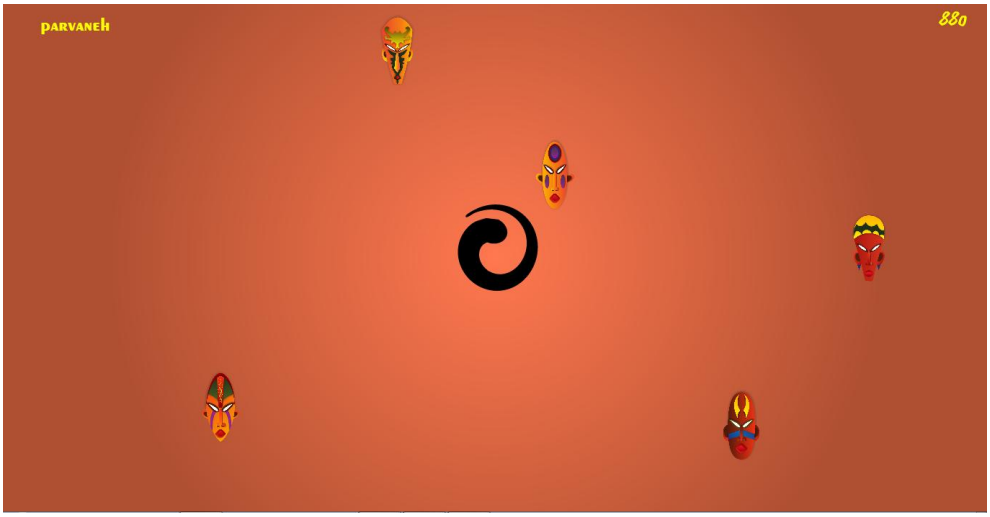
آنچه در تابع pathUpdate در هنگام به روز رسانی خط اتفاق می‌افتد به این شکل است:

```
if(mode=="defence"){
      if(!defenceEvolved){
        for(var a3=0;a3<pointArray.length;a3++){

segments[a3].point.x+=(defenceEvolutionVector.x+pointArray[a3].x-
segments[a3].point.x)/5;
```

```
segments[a3].point.y+=(defenceEvolutionVector.y+pointArray[a3].y-
segments[a3].point.y)/5;
            }
            if(Math.abs(segments[size-1].point.x-
(defenceEvolutionVector.x+pointArray[size-1].x))<5){
                for(var a4=0;a4<pointArray.length;a4++){

segments[a4].point.x=defenceEvolutionVector.x+pointArray[a4].x;

segments[a4].point.y=defenceEvolutionVector.y+pointArray[a4].y;
                }
                defenceEvolved = true;
            }
        }else{
            path.position = eventt.point;
            path.rotate(-20);
        }
    }
```

در ابتدا با توجه به بردار تبدیلی که ساخته شده، تمام نقاط خط از محل فعلی خود به محلی که مشخص شده حرکت می‌کنند تا خط شکل از پیش تعیین شده‌ای را که در کد مشخص شده است، به خود بگیرد. اما وقتی‌که تبدیل خط انجام شد متغیر کنترلی defenceEvolved مقدار true می‌گیرد تا از این پس خط تنها به دور خود بچرخد.

در زیر تصویری از بازی در زمان دفاع خط آورده شده است:



شکل ۲۰- دفاع خط در زمان حمله ماسک‌ها

۴-۱۰- حمله ماسک‌ها

در صورتی که خط از خود در مقابل حمله ماسک‌ها دفاع نکند، ماسک‌ها با خط برخورد کرده، از امتیاز شخص کاسته خواهد شد و به خط ضربه وارد خواهند کرد که بر عملکرد بازیکن تاثیر منفی خواهد گذاشت. در صورت برخورد ماسک با خط، در نقطه برخورد، خط دچار خمیدگی به شکل سهمی درجه دوم می‌شود و در اثر ضربه وارد شده بر خط، خط به صورت خمیده، در جهت حمله ماسک به گوشه canvas پرتاب می‌شود و در صورت برخورد با یکی از نقاط کناری صفحه، با رسم منحنی bezier درجه سوم، به نقطه‌ای که نشانه‌گر در آنجا قرار دارد، باز می‌گردد. در واقع، نقطه ابتدایی منحنی برابر نقطه برخورد منحنی با مرز صفحه، نقطه انتهایی منحنی برابر موقعیت نشانه‌گر و نقاط کنترلی به شیوه‌ای که قبلا گفته شد، مشخص می‌شوند. با بازگشت خط به موقعیت نشانه‌گر، بازی از سر گرفته می‌شود.

نحوه ضربه خوردن خط از برخورد ماسک، ایجاد سهمی درجه دوم، پرتاب شدن خط و بازگشت خط به موقعیت نشانه‌گر بر روی منحنی bezier درجه سوم، به صورت زیر پیاده‌سازی شده است:

ابتدا در هنگامی که یکی از ماسک‌ها به خط برخورد می‌کند، چک می‌شود که متغیر حالت خط چه مقداری دارد و اگر این مقدار normal بود، محاسبات مربوط به حمله به شکل زیر انجام می‌شود:

```
var vector=new Point();
            vector.x= this.element.position.x-segments[0].point.x;
            vector.y= this.element.position.y-segments[0].point.y;
            this.vanishVector= vector;
            return false;
          }else{
            attackVector.x = this.element.position.x - this.previousPosition.x;
            attackVector.y = this.element.position.y - this.previousPosition.y;
            attackVector.length = 12;


Point(this.element.position.x+attackVector.x,this.element.position.y+attackVec
tor.y));
            var index = masks.indexOf(this);
            masks.splice(index,1);
            this.element.remove();
            attackedEvolutionShape = new Path();
            var attackVector2 = new Point(attackVector.x,attackVector.y);
            var  pathHitIndex = (hitResult.segment.index<size-
2)?(hitResult.segment.index+1):(2*(size-2)-hitResult.segment.index);
            for(var i=pathHitIndex;i<size;i++){
               var temp = (i-pathHitIndex)*(i-pathHitIndex);
               attackVector2.length = Math.abs(100-temp)/3.5;
               if(temp<100){
```

```
                attackedEvolutionShape.add(new Point(segments[i].point.x + attackVector2.x,segments[i].point.y + attackVector2.y));
            }else{
                attackedEvolutionShape.add(new Point(segments[i].point.x - attackVector2.x,segments[i].point.y - attackVector2.y));
            }
        }
        for(var j=pathHitIndex-1;j>=0;j--){
            var temp2 = (pathHitIndex-j)*(pathHitIndex-j);
            attackVector2.length = Math.abs(100-temp2)/3.5;
            if(temp2<100){
                attackedEvolutionShape.insert(0,new Point(segments[j].point.x + attackVector2.x,segments[j].point.y + attackVector2.y));
            }else{
                attackedEvolutionShape.insert(0,new Point(segments[j].point.x - attackVector2.x,segments[j].point.y - attackVector2.y));
            }
        }
        attackedEvolutionShape.strokeColor = 'blue';
        attackedEvolutionVectorsToShape = new Array();
        for(var k=0;k<size;k++){
            attackedEvolutionVectorsToShape.push(new Point(attackedEvolutionShape.segments[k].point.x-segments[k].point.x,attackedEvolutionShape.segments[k].point.y-segments[k].point.y));
            attackedEvolutionVectorsToShape[k].length /=5;
        }
        mode = "attacked";
        attackedEvolved = false;
```

در ابتدا جهت حمله شده به خط با به‌دست آوردن مختصات برداری از محل قبلی ماسک تا محل جدید آن به‌دست می‌آید. سپس ماسک خود از بین رفته و اثر حمله خود را می‌گذارد. برای این کار نقطه‌ای از خط که برخورد ماسک با آن صورت گرفته استخراج می‌شود. آن نقطه به عنوان نقطه مرکزی یک سهمی در نظر گرفته شده و به اندازه برداری که گفته شد به عقب برده می‌شود و نقاط بعد و قبل آن با فواصل مشخص در رابطه یک سهمی جای می‌گیرند. لازم به ذکر است که این مختصات جدید نه برای خود نقاط خط که در یک آرایه به نام attackedEvolutionVectorsToShape ذخیره می‌شوند. این آرایه به همراه تغییر متغیر حالت خط به attacked کمک می‌کند که خط به شکل مورد نظر تبدیل شود.

در نتیجه، در تابع به روز رسانی خط قطعه کد زیر اجرا می‌شود:

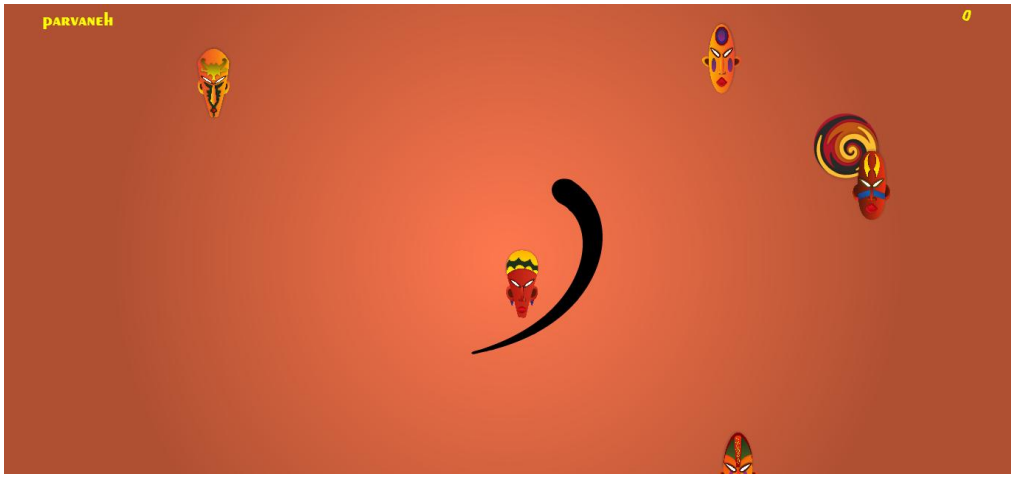
```
if(!attackedEvolved){
            var evolvedSegmentsCounter = 0;
            for(var i=0;i<size;i++){
               segments[i].point.x += (attackedEvolutionVectorsToShape[i].x);
               segments[i].point.y += (attackedEvolutionVectorsToShape[i].y);
               if(Math.abs(attackedEvolutionShape.segments[i].point.x-
segments[i].point.x)<1 &&
Math.abs(attackedEvolutionShape.segments[i].point.y-segments[i].point.y)<1)
                  evolvedSegmentsCounter++;              }
            if(evolvedSegmentsCounter==size){
               attackedEvolved = true;
            }
         }else{
            path.position.x += attackVector.x;
            path.position.y += attackVector.y;
if(path.bounds.right>x||path.bounds.left<0||path.bounds.top<0||path.bounds.bott
om>y){
               mode="attackRecovery";  }
         }
```

در این کد به مانند آنچه در هنگام دفاع رخ می‌داد، خط به شکل تعیین شده در attackedEvolutionVectorsToShape حرکت می‌کند و وقتی کاملا به آن شکل درآمد کل خط شروع به حرکت به جهتی می‌کند که در بردار حمله از حمله ماسک استخراج شده بود. به این صورت خط بعد از تغییر شکل دادن در جهتی که به آن حمله شده است حرکت می‌کند. در آخر کنترل می‌شود که آیا خط به مرزهای صفحه رسیده است یا خیر که در این صورت حالت خط به attackRecovery تبدیل می‌شود که در نتیجه آن خط شروع به حرکت به سمت جای نشانه‌گر می‌کند. کد این حرکت اینگونه است:

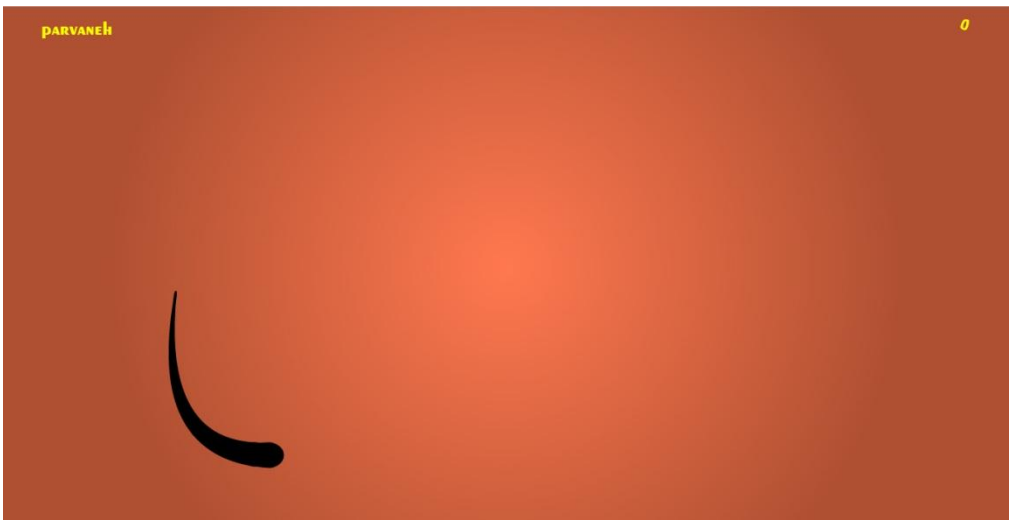
```
if(mode=="attackRecovery"){
        var p0 = segments[0].point;
        var p1 = new Point(x-100,y-100);
        var p2 = new Point(x-100,100);
        var p3 = eventt.point;
        var cx = 3*(p1.x - p0.x);
        var bx = 3*(p2.x - p1.x) - cx;
        var ax = p3.x - p0.x - cx - bx;
        var cy = 3*(p1.y - p0.y);
        var by = 3*(p2.y - p1.y) - cy;
        var ay = p3.y - p0.y - cy - by;
        var tspeed = 0.02;
        if(t<1){
           segments[0].point.x = ax*t*t*t + bx*t*t+cx*t + p0.x;
           segments[0].point.y = ay*t*t*t + by*t*t+cy*t + p0.y;
           t+=tspeed;
        }else{
           t=0;
           mode = "normal";
        }
     }
```

در واقع نقطه ابتدایی خط در طول یک منحنی Bezier که از دو نقطه کنترلی از پیش تعیین شده تشکیل شده، حرکت کرده تا به مکان نشانه‌گر برسد و سپس خط به حالت normal باز می‌گردد.

در زیر تصاویری از حمله ماسک‌ها به خط و پرتاب شدن خط به گوشه‌ای از صفحه پس از حمله آورده شده است:



شکل ۲۱- برخورد ماسک به خط در زمان حمله



شکل ۲۲- پرتاب شدن خط به گوشه صفحه پس از حمله ماسک‌ها

۴-۱۱- انیمیشن نواختن تامبورین

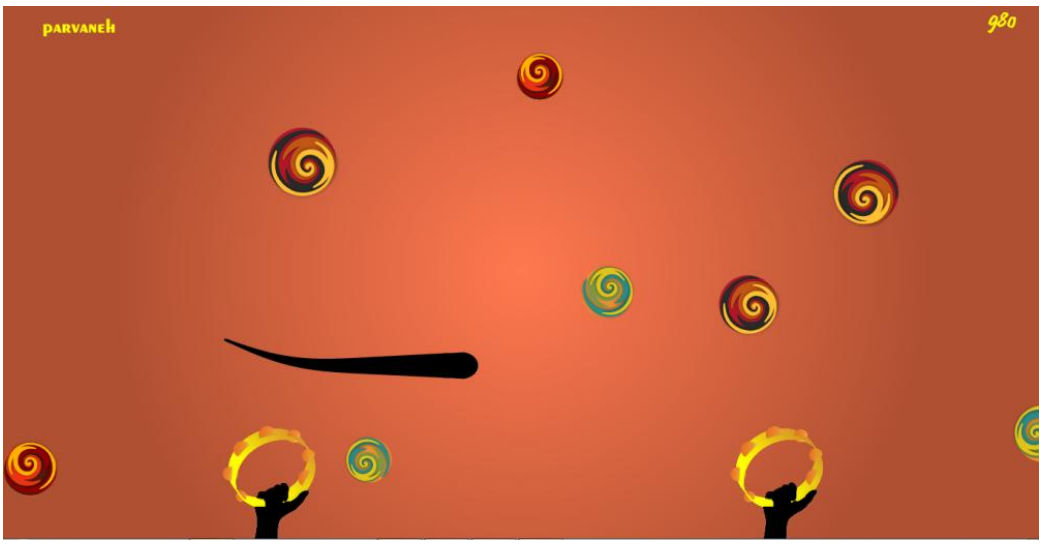
در پایان مرحله اول و با شروع مرحله دوم، در مدت زمان مشخصی، دو دست که تامبورین می‌نوازند، وارد صفحه بازی می‌شوند و انیمیشنی را به منظور تشویق شخص بازیکن اجرا می‌کنند. در زمان انجام این انیمیشن، صدای موسیقی اصلی متن بازی آهسته کم می‌شود و صدای نواختن تامبورین‌ها شنیده می‌شود و با پایین رفتن دست‌ها و اتمام انیمیشن، صدای موسیقی به حالت اولیه باز می‌گردد و بازی ادامه می‌یابد. نحوه پیاده‌سازی این انیمیشن، به صورت زیر است:

```
function playTambourine(){
    if(tambourinePosition==1){
        for (var e = 0; e < 2; e++) {
            var tambourineItem = tambourines[e];
            if(tambourineItem.position.y> y-55){
                moveTambourine();
                tambourineItem.position.y--;
            }
        }
        if(volume==1){
            music.volume-=.01;
            if(music.volume<.1)
                volume=2;
        }
        if(tambourineItem.position.y<y-55)
            tambourinePosition=2;
    }
    if(tambourinePosition==2){
        moveTambourine();
        if(music2.currentTime >3.9)
            tambourinePosition=3;
    }
```

```
    if(tambourinePosition==3){
        for (var e1 = 0; e1 < 2; e1++) {
            var tambourineItem1 = tambourines[e1];
            if(tambourineItem1.position.y< y+95){
                moveTambourine();
                tambourineItem1.position.y++;
            }
        }
        if(volume==2){
            music.volume+=.01;
            if(music.volume>.99)
                volume=0;
}
        if(tambourineItem1.position.y>y+95)
            tambourinePosition=0; }
  }
  function moveTambourine(){
    for (var e = 0; e < 2; e++) {
        var tambourineItem = tambourines[e];
        if(rotateTambourine==1){
            tambourineItem.rotate(1);
            tambourineCount++;
            if(tambourineCount==5){
                rotateTambourine=2;
                tambourineCount=0; }
        }
        if(rotateTambourine==2){
            tambourineItem.rotate(-2);
            tambourineCount++;
            if(tambourineCount==5){
                rotateTambourine=3;
                tambourineCount=0;
```

```
            }
        }
        if(rotateTambourine==3){
            tambourineItem.rotate(1);
            tambourineCount++;
            if(tambourineCount==5){
                rotateTambourine=1;
                tambourineCount=0;
            }
        }
    }
}
```

در زیر تصویری از نواختن تامبورین دیده می‌شود:



شکل ۲۳- انیمیشن نواختن تامبورین در زمان شروع مرحله دوم

۴-۱۲- جمع‌بندی و نتیجه‌گیری

در این فصل، جزئیات بازی پیاده‌سازی شده مطرح شد. ایجاد و توسعه این بازی با استفاده از ویژگی‌های جدید HTML5 و عنصر جدید canvas، نشان داد که شاید بتوان قابلیت‌های جدید این تکنولوژی را با قابلیت‌های فلش مقایسه کرد. یعنی ترسیم اشکال، ایجاد انیمیشن و همچنین ایجاد و توسعه محیط‌های تعاملی از جمله بازی. به دلیل استفاده از اسکریپت برای کار با عنصر canvas، امکان تعاملی کردن این محیط وجود دارد و می‌توان به طراحی بازی در آن پرداخت. شاید بتوان گفت، یکی از انگیزه‌های اساسی در ایجاد عنصر canvas، به وجود آوردن جایگزینی برای Flash بوده است تا بتوان علاوه بر مقابله با مشکلاتی که افزونه‌های Flash داشتند، امکانات آن‌ها را در محیط وب حفظ کرد.

نتیجه‌گیری

در این پروژه تلاش شد تا نشان داده شود چگونه تلاش‌های استانداردسازی محتوای چندرسانه‌ای در محیط وب با استفاده از تکنولوژی‌های جدید معرفی شده در HTML5 می‌توانند نویددهنده عصری جدید در عرضه این‌گونه محتوا در اینترنت باشند. مهمترین عناصر جدید در HTML5 عنصرهای canvas، audio و video است که سه نوع داده مهم در دنیای امروز وب را در خود جای می‌دهند. در این میان canvas به عنوان بستری برای ترسیم اشکال گرافیکی برداری با قابلیت مهم اسکریپت نویسی قدرت خلق انواع داده تعاملی از جمله بازی‌ها را در اختیار گذاشته است.

آن چه در نهایت هدف پیاده‌سازی در این پروژه بود، کاویدن قابلیت‌های این عنصر و نشان دادن قدرت آن در اجرای هدفش بود. همانطور که نشان داده شد، انواع رسوم گرافیکی، برداری، تصویری پویا با قابلیت‌های تعامل بدون محدودیت در محیط canvas از جمله توانایی‌های مهم این عنصر هستند که در این پروژه از تمام آن‌ها استفاده کردیم و نشان دادیم که با کمک آن‌ها می‌توان یک بازی در حد و اندازه بازی‌های دیگری که در محیط وب تا کنون وجود داشته‌اند و حتی با امکانات و خصوصیاتی بارزتر و کاراتر، ساخت و توسعه داد. این پروژه و بازی پیاده‌سازی شده، مبین این مطلب و تکنیک‌های به‌کار رفته در آن که در این گزارش نیز توضیح داده شده‌اند به عنوان شاهدی بر این موضوع است.

تجربه این پروژه با canvas در راستای شناخت بیشتر آن و در نهایت مقایسه‌اش با تکنولوژی‌های مشابه که تا قبل از آن وجود داشته (به خصوص Flash)، ما را به این نتیجه می‌رساند که علی‌رغم نبود ابزارها و کتابخانه‌های کافی برای کار با آن و ایجاد محتواهای گرافیکی و پویا، این تکنولوژی در بطن خود تمام ویژگی‌های اولیه لازم برای رقابت با تکنولوژی‌های مشابه را دارد، اما برتری مهم آن در مقایسه با سکوهایی چون Flash در واقع استاندارد شدن آن در هسته اصلی وب یعنی HTML است که این قدرت را به آن می دهد تا از مشکلات امنیتی عبور کرده و در هر سیستم کاربری به روزی به‌خوبی اجرا شود.

منابع

[1] R. Cecco, Supercharged JavaScript Graphics: with HTML5 canvas, jQuery, and More, O'Reilly Media; first edition, 2011.

[2] B. McLaughlin, What Is HTML5?, O'Reilly Media; first edition, 2011.

[3] J. F. Steve Fulton, HTML5 Canvas, O'Reilly Media;, 2011.

[4] D. Geary, Core HTML5 Canvas: Graphics, Animation, and Game Development (Core Series), Prentice Hall, 2012.

[5] M. Pilgrim, HTML5: Up and Running, O'Reilly Media, 2010.

[6] K. Simpson, JavaScript and HTML5 Now, O'Reilly Media, 2012.

[7] "2D Context Draft," 17 12 2012. [Online]. Available: http://www.w3.org/TR/2012/CR-2dcontext-20121217/.

[8] "HTML5 Draft," 17 12 2012. [Online]. Available: http://www.w3.org/TR/2012/CR-HTML5-20121217/.

[9] "Mozilla HTML5," [Online]. Available: https://developer.mozilla.org/en-US/docs/HTML/HTML5.

[10] "Mozilla Canvas Guide," [Online]. Available: https://developer.mozilla.org/en-US/docs/HTML/Canvas/Tutorial?redirectlocale=en-US&redirectslug=Canvas_tutorial.

[11] "Mozilla Canvas Element," [Online]. Available: https://developer.mozilla.org/en-US/docs/HTML/Element/canvas.