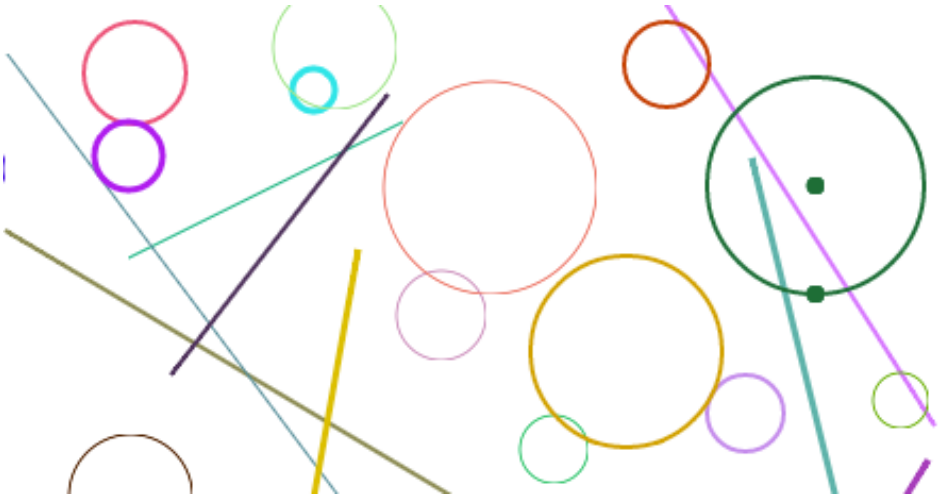
Computergrafik 2 / Aufgabe 1.1

JavaScript und 2D Canvas



Lernziele / Motivation

In dieser Aufgabe üben Sie das Lesen und Verstehen von fremdem JavaScript-Code und bauen einige Objekte und Funktionen für interaktive 2D-Grafik im HTML `<canvas>`-Element.

Vorbereitung

Laden Sie `cg2-01-canvas.zip` von Moodle herunter und entpacken Sie dieses Verzeichnis, so dass es auf der gleichen Ebene wie `cg2-00-warmup/` und `lib/` aus der Warmup-Übung zu liegen kommt. Öffnen Sie die darin enthaltene `index.html` mit dem Browser.

Die Applikation erlaubt dem User, über einen Button "new line" neue Liniensegmente (mit zufälligen Eigenschaften) zur Szene hinzuzufügen. Durch Klicken auf ein Liniensegment wird dieses ausgewählt und kann dann mittels Anfassern (*Draggers*) manipuliert werden.

Das Framework für Aufgabe 1 besteht aus einer HTML-Datei und einigen JavaScript-Modulen. Studieren Sie den Code (beginnend bei `main.js`) und machen Sie sich mit dem Aufbau und der Funktionsweise dieser Module vertraut. Versuchen Sie, die in der SU vorgestellten Patterns zu erkennen und nachzuvollziehen. *Mehr zum Aufbau des Frameworks finden Sie im Anhang dieses Aufgabenblatts.*

Aufgabenstellung

1. Fügen Sie der `index.html` einen weiteren Button hinzu, mit dem neue Kreise mit zufälliger Position und Größe erzeugt werden können. Implementieren Sie dazu in `html_controller.js` einen neuen Callback, der beim Klicken des neuen Buttons aufgerufen wird und ein `Circle`-Objekt erzeugt (siehe 2).

2. Implementieren Sie analog zu `straight_line` ein Modul `circle`, welches interaktiv manipulierbare Kreise für die Szene bereitstellt. Objekte vom Typ `Circle` benötigen folgende Methoden:
   a. Konstruktor (mit Mittelpunkt und Radius als Parameter)
   b. `draw(context)` stellt den Kreis in dem als Parameter übergebenen 2D Rendering-Kontext grafisch dar. Verwenden Sie z.B. die Canvas-2D-Funktion `arc()`, ein Beispiel finden Sie in `PointDragger.draw()`.
   c. `isHit(pos)` überprüft, ob die als Parameter übergebene Mausposition auf den Kreis fällt. (Vgl. auch `PointDragger.isHit()`). Hierbei soll nicht getestet werden, ob die Maus sich im Inneren des Kreises befindet, sondern ob der Rand des Kreises getroffen wurde. Implementieren Sie eine kleine Toleranz, z.B. von +/-2 Pixeln.
   d. `createDraggers()` erzeugt eine Liste von Anfasser-Objekten, mit denen man den Kreis interaktiv manipulieren kann. Verwenden Sie zunächst einen einfachen `PointDragger`, um den Mittelpunkt des Kreises verschiebbar zu machen.
   e. Implementieren Sie einen weiteren Dragger, der es dem Benutzer erlaubt, den Radius des Kreises zu manipulieren. Implementieren Sie dazu ggf. einen neuen Dragger-Typen, der seine Position abhängig von der Position und dem Radius des Kreises verändert.

3. Fügen Sie der HTML-Seite einen Parameter-Bereich hinzu, in welchem die Farbe und die Liniendicke des gerade ausgewählten Elements manipuliert werden können. Verwenden Sie dazu z.B. HTML-`<input>`-Elemente vom Typ "color" bzw. "number" [1]. Erweitern Sie das Modul `html_controller` um die entsprechende Funktionalität. Beim Selektieren und Manipulieren eines Objekts werden die aktuellen Werte des Objekts angezeigt, und beim Verändern der Werte in den Eingabefeldern wird das gerade ausgewählte Objekt im Canvas aktualisiert dargestellt. Tipp: Mittels `SceneController.onSelection()` und `SceneController.onObjectChange()` können Sie eine Callback-Funktion registrieren, die aufgerufen wird, wenn sich die Selektion oder das Objekt ändern; mittels `SceneController.getSelectedObject()` können Sie das aktuell selektierte Objekt abfragen.

4. Erweitern Sie den Parameter-Bereich der HTML-Seite so, dass Sie bei Kreisen auch der Radius angezeigt wird und durch Eingabe einer Zahl verändert werden kann. Bei Objekten, die keine Radius-Eigenschaft besitzen, soll auch kein Radius angezeigt werden. Tipp: jQuery `show()` und `hide()`.

[1] siehe z.B. `http://www.w3.org/TR/html-markup/input.color.html` und `http://www.w3.org/TR/html-markup/input.number.html`

Implementierungshinweise

Verwenden Sie wo sinnvoll die in der SU vorgestellten Konzepte von Modulen, Objekten und Callbacks. Entwerfen Sie den Code objektorientiert und passend zum Framework. Verwenden Sie keine zusätzlichen externen Bibliotheken. Vermeiden Sie jegliche Verunreinigung des globalen Namensraums. Dokumentieren Sie den Code ungefähr so ausführlich wie auch im Framework vorgegeben.

Abgabe

Dies ist Aufgabe 1.1; die Bearbeitungszeit der Aufgabe ist für ca. eine Woche ausgelegt, kann aber aufgrund der Anfangshürden etwas mehr Zeit benötigen. Die Abgabe der gesamten Aufgabe 1 (1.1, 1.2, 1.3) soll via Moodle bis zu dem dort angegebenen Termin erfolgen. Verspätete Abgaben werden wie in den Handouts beschrieben mit einem Abschlag von 2/3-Note je angefangener Woche Verspätung belegt. Geben Sie bitte pro Gruppe jeweils nur eine einzige `.zip`-Datei mit den Quellen Ihrer Lösung sowie mit den ggf. geforderten Screenshots ab.

Demonstrieren und erläutern Sie dem Übungsleiter Ihre Lösung *in der nächsten Übung nach dem Abgabetag*. Die Qualität Ihrer Demonstration ist, neben dem abgegebenen Code, ausschlaggebend für die Bewertung! Es wird erwartet, dass alle Mitglieder einer Gruppe anwesend sind und Fragen beantworten können.

Anhang: Struktur des Mini-Frameworks

Die Datei `index.html` definiert u.a. einen HTML5-Canvas, zu welchem in `main.js` ein 2D-Rendering-Kontext erzeugt wird.

Des weiteren wird eine initial leere Szene (`scene.js`) erzeugt, welche eine einfache Sammlung von Objekten darstellt, die jeweils über eine `draw()`-Methode verfügen müssen, um sich im Canvas darzustellen. Mittels `Scene.addObjects()` können neue Objekte zur Szene hinzugefügt werden (Vorsicht: `add()` erwartet ein Array von Objekten); dabei werden später hinzugefügte Objekte auch später gemalt; d.h. das zuerst hinzugefügte Objekt liegt "hinten" und das zuletzt hinzugefügte "vorne." Die Szene unterscheidet nicht zwischen "echten" Objekten und Anfassern - es behandelt alle Objekte gleich.

Ein `SceneController` kontrolliert die Applikationslogik bei der Interaktion des Benutzers mit dem Canvas; er ist der Vermittler zwischen dem Canvas als "View" und den Szenen-Objekten als "Modell". Er stellt einen Mechanismus zu Verfügung, der Maus-Events im Canvas-Element abfängt und diese an die Szenen-Objekte verteilt. Dazu werden alle Objekte in der Reihenfolge "front-to-back" durchlaufen, und für jedes Objekt wird mittels der Methode `isHit()` ermittelt, ob die Mausposition dieses Objekt trifft. Dem Objekt, welches beim Herunterdrücken der Maustaste getroffen wurde, werden dann z.B. die Drag-Events zugestellt. Des Weiteren implementiert der

`SceneController` die Auswahl-Logik: wann wird welches Objekt selektiert / deselektiert.

Der `HtmlController` definiert Event-Handler für alle HTML-Elemente außer dem Canvas. Z.B. wird hier die Funktion definiert, die beim Klicken auf "new line" aufgerufen wird und ein Objekt vom Typ `StraightLine` zur Szene hinzufügt, mit zufälligen Koordinaten und Farben.

`StraightLine`: Neben der `draw()`-Methode definiert eine Linie zwei Anfasser, mit denen die Endpunkte interaktiv positioniert werden können (`PointDragger`). Dabei sind die Anfasser vom Design her von der Linie unabhängig; sie erhalten im Konstruktor Callback-Funktionen zum Lesen und Setzen der Position, über welche sie mit den ihnen zugeordneten Objekten kommunizieren. Auch die Anfasser sind Szenen-Objekte, die sich selbst mittels `draw()` im Canvas darstellen können.

2D-Vektoren: An vielen Stellen im Code werden 2D-Punkte oder -Vektoren als einfaches Array mit zwei Werten `[x,y]` verwendet. Das Modul `vec2.js` bietet einige rudimentäre Funktionen zum Rechnen mit 2D-Vektoren.

Das Framework ist im Code relativ ausführlich dokumentiert - Fragen können natürlich in der Übung, im Moodle-Forum und im Tutorium gestellt werden!