

Élaboration d'un jeu multijoueur en HTML5

Florent Marchand de Kerchove

Faculté des sciences et techniques
Université du Havre

27 juin 2011

Introduction

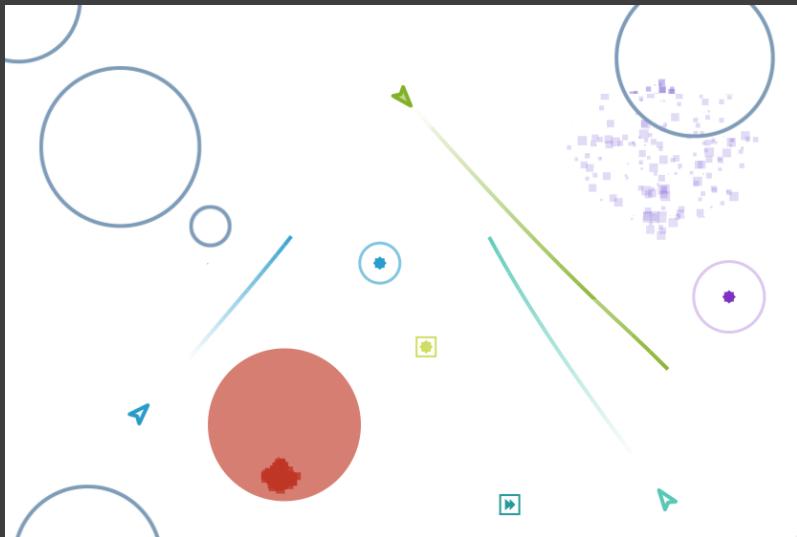


Table des matières

1 Technologies fondatrices

JavaScript

Canvas HTML

WebSocket

Node.js

2 Derrière Spacewar

Principe de jeu

Côté client

Côté serveur

1 Technologies fondatrices

JavaScript

Canvas HTML

WebSocket

Node.js

2 Derrière Spacewar

Principe de jeu

Côté client

Côté serveur

JavaScript

Historique

- Langage de programmation des navigateurs web
- Créé en 1995 par Brendan Eich chez Netscape
- Standardisé sous le nom d'ECMAScript en 1996
- Initialement limité, regain d'intérêt avec AJAX
- Nombreux frameworks puissants (jQuery, CommonJS, Dojo)
- Présent sur la majorité des sites d'aujourd'hui

JavaScript

Caractéristiques

- Descend de Scheme et Self
- Langage fonctionnel (fonctions de première classe, *closures*)
- Langage dynamique (faiblement typé, évalué à l'exécution)
- Héritage par prototype
- Syntaxe empruntée à Java

Avantages :

- Langage de haut niveau
- Multi-plateforme (requiert un navigateur)
- Fonctionnalités asynchrones
- Programmation événementielle

CoffeeScript : langage intermédiaire plus épuré

Canvas HTML

Description

Élément `<canvas>` :

- Permet de dessiner et d'animer librement sur une page web
- Contextes 2d et 3d (WebGL)
- API stable
- Implémenté dans les navigateurs majeurs

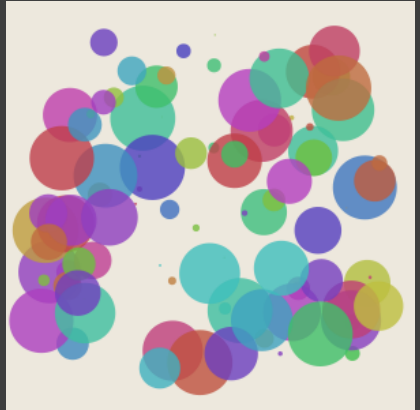
Alternative aux SVG plus performante :

- Surface *bitmapped* plutôt que vectorielle
- Pas d'insertion dans le DOM
- Accélération matérielle possible

Canvas HTML

Exemple

```
function init() {  
  var canvas =  
    document.querySelector('#canvas');  
  var ctx = canvas.getContext('2d');  
  ctx.fillStyle = 'hsl(40, 30%, 90%)';  
  ctx.fillRect(0,0,300,300);  
  
  for (var i=0; i < 10; ++i) {  
    ctx.save();  
    ctx.translate(Math.random()*250,  
                  Math.random()*250);  
    ctx.fillStyle = 'hsla(' +  
      Math.random()*360 +  
      ', 50%, 50%, 0.8)';  
    ctx.beginPath();  
    ctx.arc(24, 24, 12 + 12*Math.sin(i),  
            Math.PI*2, false);  
    ctx.fill();  
    ctx.restore();  
  }  
}
```



Protocole WebSocket

Description

- Réponse du standard aux techniques Comet
- Rend obsolète HTTP *long-polling* et HTTP *streaming*
- Véritable *full-duplex* entre client et serveur HTTP
- Mise à jour de la connexion TCP créée pour la requête HTTP
- Protocole en évolution

Node.js

Description

- Serveur performant écrit en JavaScript
- Fournit une API réseau élémentaire
- Utilise le moteur JavaScript V8 de Google
- Entrées/sorties asynchrones (*epoll*, *kqueue*, ...)
- Programmation événementielle
- Nombreux modules dont Socket.IO pour WebSocket

Serveur écho

```
var net = require('net');  
  
var server = net.createServer(function (socket) {  
  socket.write("Echo server");  
  socket.pipe(socket);  
});
```

Node.js

Exemples

Serveur http

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World');
}).listen(1337, "127.0.0.1");
```

① Technologies fondatrices

JavaScript

Canvas HTML

WebSocket

Node.js

② Derrière Spacewar

Principe de jeu

Côté client

Côté serveur

Principe de jeu

Démonstration

Principe de Spacewar :

- Jeu d'action frénétique dans l'espace
- À chaque joueur un vaisseau
- But : tirer sur les autres et survivre

► Démonstration

Principe de jeu

Éléments du jeu

- Contrôles simples :
 - Tourner à gauche, à droite
 - Avancer
 - Tirer
 - Utiliser un bonus
- Carte torique
- Obstacles : planètes et satellites
- Trajectoire des balles affectées par la gravité des planètes
- Les bonus apportent de la variété (mines, turbo, bouclier, ...)

Client

Rôle du client

- Relayer les entrées claviers au serveur
- Recevoir les messages du serveur
- Afficher le jeu en temps réel

Semblable à un terminal : toute la logique est côté serveur.

Client

Boucle de dessin

- Un jeu d'action exige un rendu fluide (40 à 60 FPS)
- Requiert de dessiner très rapidement une frame
- Utilise uniquement le canvas HTML

```
redraw = (context) ->
    context.clearCanvas()

    centerView()

    for obj in gameObjects
        obj.draw(context) if obj.inView()

    for e in effects
        e.draw(context) if e.inView()

    drawInfinity(context)

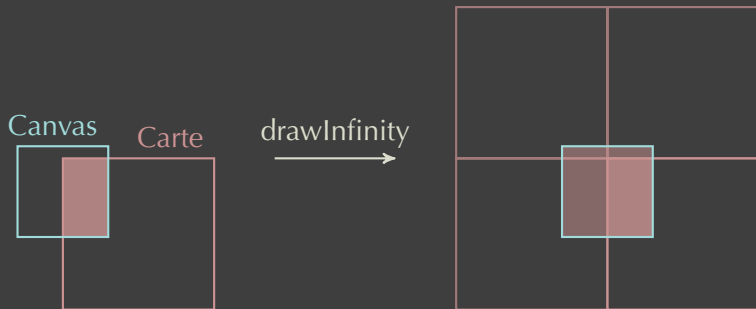
    drawUI(context)
```

Client

Dessiner le tore

Donner l'illusion d'une carte torique :

- Remplir le canvas de copies de la carte
- Considérer les entités les plus proches sur le tore
- Appliquer la logique de jeu au tore côté serveur

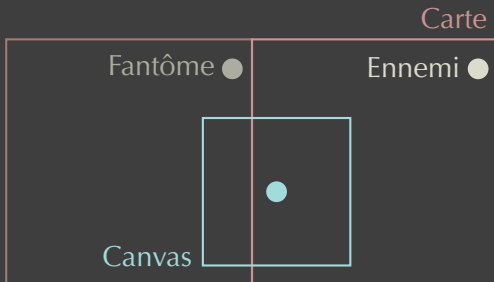


Client

Dessiner le tore

Donner l'illusion d'une carte torique :

- Remplir le canvas de copies de la carte
- Considérer les entités les plus proches sur le tore
- Appliquer la logique de jeu au tore côté serveur



Client

Performance

Comment optimiser le dessin sur le client ?

- Ne pas dessiner les objets hors champ
- Sauvegarder les dessins coûteux dans des *sprites*
- Optimisations de bas niveau hors de notre contrôle
- Accélération matérielle fournie par les navigateurs

Serveur

Rôle du serveur

- Gérer la logique du jeu :
 - Initialiser la carte de jeu
 - Mouvoir les objets (vaisseaux, planètes, balles, ...)
 - Détecter les collisions entre objets
 - Résoudre ces collisions
- Synchroniser l'information auprès des clients

Serveur

Communications clients-serveur

Connexion d'un client :

- Attribution d'un identifiant
- Création d'un objet *Player* associé
- Envoi de tous les objets de jeu

Durant la partie :

- Les clients envoient leurs entrées clavier
- Le serveur broadcast les changements

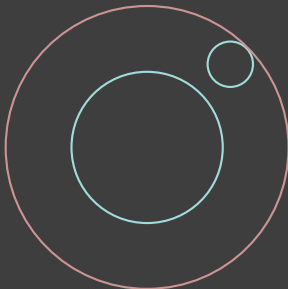
Déconnexion d'un client :

- Notification aux autres clients
- Libération des ressources associées

Serveur

Initialiser la carte de jeu

- Chargement du fichier de préférences :
 - Dimensions de la carte
 - Nombre de planètes à placer
 - Taille des planètes, des satellites
 - Vitesse et distance des satellites
- Placement aléatoire sans chevauchements



Serveur

Boucle principale

Étapes effectuées toutes les 20ms :

- Agir en fonction des évènements clavier
- Déplacer tous les objets
- Détecter et résoudre les collisions
- Récolter les changements d'état de chaque objet
- Diffuser les changements de tous les objets

Serveur

Gérer les collisions

Traitement symétrique centralisé :

```
'ship-mine': (ship, mine) ->  
  ship.explode()  
  mine.nextState() if mine.state is 'active'
```

Algorithme performant crucial :

- Approche naïve quadratique
- Vérification des collisions entre voisins
- Découpage de la carte en grille

Améliorations envisagées

- Instanciation des parties :
 - Rejoindre une partie aléatoire ou entre amis
 - Création de parties personnalisées
- Communication entre joueurs
- Optimisations serveur :
 - Diminuer le coût des collisions
 - Permettre un plus grand nombre de joueur simultanés

Améliorations envisagées (2)

- Optimisations client :
 - Dessiner plus rapidement
 - Améliorer la compatibilité avec tous les navigateurs
- Éléments de jeu supplémentaires :
 - Bonus (bouclier, missile)
 - Contenu solo
 - Mesure de progrès (score, statistiques)

Autant de prétextes pour apprendre

Merci

Questions / Réponses