# Examen ProgWeb - IM 47

## Conditions d'examen

Le travail est à rendre par email à nicolas.chabloz@heig-vd.ch **avant 16h16** avec en pièce jointe un *zip* de tout votre projet, sauf le répertoire **node_modules**. (Vous êtes responsable de son contenu, et devez-vous assurer de la bonne réception de celui-ci en fin d'examen.)

- Tout document papier (et électronique en local ou sur le site du cours) autorisé.
- Utilisation de l'ordinateur restreinte aux applications suivantes: éditeur de code, browser restreint aux sites Web du cours et des liens qui s'y trouvent (mais aucun autre site, ni accès aux forums des sites).
- Pas d'autre communication autorisée pendant le travail.

**Le non-respect de ces conditions donnera la note de 1 et l'impossibilité de se présenter à l'examen de remédiation !**

**Mise en place (5 minutes)**

Vous devez développer un jeu d'adresse demandant au joueur d'atteindre le plus rapidement des cibles (démo en début du cours). Commencez par créez un fichier nommé *index_shooter.js* dans votre répertoire *src*, ainsi qu'un autre nommé *index_shooter.html* dans *dist*. Vous pouvez reprendre le code HTML du TP sur l'effet parallaxe. N'oubliez pas aussi d'y changer le lien vers le fichier JS qui se nommera *bundle_shooter.js*. Finalement, lancez les deux outils nécessaires à votre développement: *live-server* (**optionnel**) et *webpack* en mode *watch* (**indispensable**). Vérifiez que tout se passe bien avec un *Hello World*.

**Boucle d'animation et *canvas* (5 minutes)**

Mettez en place une boucle d'animation en utilisant la méthode *requestAnimationFrame* ou en utilisant *mainloop.js*. Cette boucle doit pouvoir s'arrêter à la fin du jeu (plus facile avec mainloop.js). Votre méthode de gestion des *frames* devra être capable de calculer le $\Delta t$ avec la *frame* précédente, ainsi que le temps total écoulé depuis le début de l'animation. Récupérez ensuite le contexte *2d* du canvas dans votre code principal, et modifiez sa taille pour qu'elle soit égale à celle de son élément DOM.

**Classe *Shooter*, constructeur ( 10 minutes)**

Créez une classe *Shooter* dans le dossier */class* dans vos sources. Ajoutez un constructeur. Cette classe représentera le "canon" du joueur. Les propriétés suivantes seront utilisées: **x, y** (pour sa position dans le plan), **r** (pour le rayon du cercle qui représentera le canon), **turnSpeed** (la vitesse de rotation fixée par défaut à 0.003 radian par ms), **fireRate** (la fréquence de tir en projectile par seconde), **angle** (l'angle de tir en radian valant *Math.PI / 2* par défaut) et finalement **color** (avec une valeur de votre choix par défaut pour la couleur).

**Méthode *draw*, dessin du cercle (10 minutes)**

Ajoutez une méthode pour le dessin du canon dans votre classe. Le contexte graphique sera reçu en paramètre. Le canon sera simplement représenté par un demi-cercle (ou un cercle entier, si vous le voulez, puisque l'on ne verra pas sa moitié inférieure). Dessinez donc un cercle remplit avec la bonne couleur dans votre méthode.

Testez votre classe en créant un "canon" dans votre programme principal et en le dessinant dans votre boucle d'animation. Le centre du cercle sera au milieu de la largeur du *canvas*, et tout en bas du *canvas*. Son rayon sera de 30px.

#### Méthode *draw*, ajout du canon (20 minutes)

Pour le canon, utilisez une simple ligne avec une largeur 5px. Voici le code permettant de dessinez une ligne de cette taille:

```
ctx.beginPath();
ctx.strokeStyle = color;
ctx.lineWidth = 5;
ctx.moveTo(x1, y1);
ctx.lineTo(x2, y2);
ctx.closePath();
ctx.fill();
ctx.stroke();
```

color représente la couleur de la ligne. Vous pouvez prendre la même que la couleur du cercle. x1 et y1 représentent le début de la ligne (prenez le centre du cercle pour ces valeurs), et x2, y2 représentent la fin de la ligne. Comme le joueur pourra faire tourner le canon, il va vous falloir calculer x2 et y2 grâce à l'angle de tir. Vous pouvez donc calculer ces valeurs grâce aux formules suivantes (qu'il vous faudra *traduire* en JavaScript):

```
x2 = x1 + 2 * r * cos(angle)
y2 = y1 + 2 * r * sin(angle)
```

Rajoutez le code nécessaire au dessin du canon dans votre méthode *draw* et testez l'affichage du canon.

### Commande de rotation du canon par le joueur (15 minutes)

Pour que le joueur puisse tourner le canon, vous allez tester si les touches A (tourner à gauche), ou D (tourner à droite) sont pressées. (De la même manière que le TP sur le parallaxe, ceci sera fait dans boucle d'animation.) Si une des touches de changement de direction est appuyée, modifiez la direction du canon via l'appel d'une méthode. Cette méthode devra recevoir en paramètre le Δt et modifiera la direction du canon (rappel: qui est un angle en radian) en y ajoutant (si on tourne à droite) ou soustrayant (si on tourne à gauche) une valeur calculée grâce au Δt et à la vitesse de rotation disponible via la propriété *turnSpeed*. Il vous faudra **vérifier** que la valeur de l'angle restera bien entre **π** et **2*π** pour que le canon ne se retrouve pas en dehors de l'écran (en dessous plus précisément). Testez que tout fonctionne.

#### Projectiles, classe Bullet (10 minutes)

Créez une classe *Bullet* héritant de la classe *Cirlce* (fournie dans les corrections des TP). Son constructeur recevra les paramètres suivants: **x**, **y**, **angle**, **r** (fixé par défaut à 3), **speed** (fixé par défaut à 0.5) et **color** (fixé par défaut à *null*). Elle appellera le constructeur parent et s'occupera d'affecter l'angle. Puis surchargez la méthode *move*, pour que celle-ci ne prenne en paramètre que le Δt et qu'elle calcule correctement la nouvelle position du projectile grâce au cosinus et sinus de l'angle.

#### Projectiles, tirs et dessin (40 minutes)

Pour que le joueur puisse tirer, vous allez tester si la touche " " (espace) est pressée dans votre boucle d'animation. Si c'est le cas, appelez la méthode **fire** de votre canon. Cette méthode recevra le temps écoulé depuis le début de l'animation. Elle créera alors un nouveau projectile grâce à la classe *Bullet* créée au point précédent. Sa position sera la même que le bout de votre canon (c.f. x2 et y2 précédemment expliqué), son angle sera celui du canon, et le reste prendra les valeurs par défaut de votre classe *Bullet*. Il vous faudra alors stocker ce nouveau projectile (puisque vous en aurez besoin pour le dessiner et pour tester les collisions). Pour ce faire, vous pouvez utiliser une Map (conseillé) ou un tableau. Rajoutez donc cette structure de donnée à votre constructeur (elle sera vide par défaut), et ajoutez-y le nouveau projectile dans la méthode **fire**.

Il vous faut aussi modifier votre méthode **draw** pour que celle-ci dessine tous les projectiles (en appelant la méthode **draw** de chacun des projectiles).

Bien sûr, il faudra aussi que vos projectiles bougent ! Pour ce faire, rajoutez une méthode **updateBullets** dans votre classe *Shooter*. Elle prendra en paramètre le Δt et appellera la méthode **move** de chacun des projectiles. N'oubliez pas d'ajouter l'appel de cette méthode dans votre boucle d'animation et tester que tout fonctionne.

### Projectiles, fréquence de tir (10 minutes)

Comme vous pouvez le constater, la cadence de tir est trop grande (bien que l'effet graphique soit joli, il faut corriger cela). Trouvez un mécanisme permettant de limiter la création des projectiles afin de respecter la propriété *fireRate* de votre classe. **Indication**: utilisez le temps totale d'animation et calculer une différence de temps avec le temps du dernier tir, puis comparer le avec la fréquence de tir (*fireRate*).

### Génération des cibles (20 minutes)

Toutes les secondes, une nouvelle cible doit apparaître. Celles-ci sont simplement des cercles et vous pouvez donc utiliser la classe *Circle*. Chaque nouveau cercle aura un rayon de 20px et sa position sera en haut de l'écran (donc *y* à 0 ) et à une position *x* aléatoire entre 0 et la largeur de l'écran. La vitesse sera fixée à 0.1 et la couleur sera aléatoire. Dans votre programme principale, ajouter donc les nouveaux cercles créés chaque seconde à une *Map* (conseillé) ou un tableau. Puis faites les bouger (l'angle de mouvement sera toujours vers le bas, c.à.d $\pi / 2$) et dessinez les en appelant respectivement les méthodes **move** et **draw** de chacune des cibles aux bons endroits de votre code. Faites aussi que votre boucle d'animation s'arrête lorsqu'une des cibles atteins le bas de l'écran (vous n'avez pas besoin d'afficher un texte de fin de partie)

### Gestion des collisions entre les projectiles et les cibles (20 minutes)

Il faut supprimer les cibles touchées. Dans votre boucle d'animation, rajoutez l'appelle à la méthode **checkCollisions** de votre classe **Shooter** en fournissant en paramètre la structure contenant toutes vos cibles. La méthode **checkCollission** devra alors parcourir tous les projectiles, puis pour chacun, toutes les cibles afin d'effectuer un test de collision grâce à la méthode **isCollindingWith** de la classe *Circle*. Si vous détecter une collision, il vous faudra supprimer la cible de la structure des cibles, ainsi que le projectile de la structure des projectiles.

### Gestion des cibles bonus (15 minutes)

Toute les 15 secondes, une cible bonus doit apparaître. Procédez donc comme la création des cibles standards, mais faites que cette cible soit plus grosse (40px de rayon).
Modifiez ensuite votre méthode **checkCollission** pour qu'en cas de collision avec une cible avec un rayon de 40px cela déclenche un tir bonus.

Le tir bonus est simplement la création d'une pléthore de projectiles sur un arc de cercle. Pour faire l'effet, parcourez les radians entre **π** et **2*π** par pas de 0.05 radian et créez un projectile à la position du centre du canon avec cet angle. Pour un effet plus intéressant, créez-les avec une vitesse de 0.75 .