Examen ProgWeb - IM 46

Conditions d'examen

Le travail est à rendre par email à nicolas.chabloz@heig-vd.ch **avant 16h30** 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 (sauf les appels au WebService) 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 min 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_game.js* dans votre répertoire *src*, ainsi qu'un autre nommé *index_game.html* dans *dist*. Vous pouvez reprendre le code HTML du TP sur l'effet parallaxe, en y changeant la couleur de fond du *canvas* en blanc dans la *css*. N'oubliez pas aussi d'y changer le lien vers le fichier JS qui se nommera *bundle_game.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*.

Développement

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

Mettez en place une boucle d'animation en utilisant la méthode *requestAnimationFrame*. Cette boucle doit pouvoir se mettre en pause (mais pas par l'utilisateur, ne codez donc pas de gestion de la touche P) . Votre méthode de gestion des *frames* devra être capable de calculer le Δt avec la *frame* précédente, ainsi que le temps total d'animation. Les Δt plus grand que 34 [ms] doivent provoquer un *drop* de la *frame* en cours de traitement.

Récupérez le contexte *2d* du canvas, et modifiez sa taille pour qu'elle soit égal à celle de son élément DOM.

Classe *Spaceship*, constructeur et méthode *draw* (20 minutes)

Créez une classe *Spaceship* dans le dossier */class* dans vos sources. Ajoutez un constructeur. Les vaisseaux seront dans un plan (cartésien à deux dimensions). Ils auront les propriétés suivantes: *x*, *y*, *size*, *direction*,

speed, *maxSpeed*, *acceleration*, *turnSpeed* et *friction*. Comme vous pouvez le deviner d'après le nom des propriétés, le vecteur de vitesse sera représenté par un angle en radian (*direction*) et par la vitesse (*speed*). Les autres propriétés seront détaillées dans les points suivants, mais elles sont assez explicites. Voilà la signature du constructeur que vous pouvez reprendre pour obtenir de bonnes valeurs par défaut des propriétés:

```
constructor(x, y, size, dir = 0, acceleration = 0.0005,
turnSpeed = 0.005, friction = 0.0001, speed = 0, maxSpeed = 0.5)
```

Méthode *draw*

Ajoutez une méthode pour le dessin du vaisseau dans votre classe. Afin d'éviter de devoir dessiner un vaisseau, nous allons utilisez le caractère UTF-8 suivant: 🚀. Pour dessiner votre vaisseau, il vous suffit donc d'effectuer une translation du *canvas* en (x,y), ainsi qu'une rotation égal à l'angle de direction du vaisseau (propriété *direction*) et d'écrire le caractère 🚀 sur votre canvas (**d'une manière très proche de ce qui est fait dans le TP sur la recherche de chemin pour les flèches du flot**). La taille du caractère en [px] sera celle de la propriété *size*. **Remarque**: le caractère 🚀 étant déjà incliné de -45° (- Pi/4 en radian), n'oubliez pas de le prendre en compte dans votre rotation. Enfin, n'oubliez pas réinitialiser le canvas avec la méthode *setTransform(1, 0, 0, 1, 0, 0)*.

Testez votre classe en créant un vaisseau et en le dessinant dans votre boucle d'animation. Sa position initiale sera au centre du *canvas*, sa taille sera de 30px et il sera dirigé vers l'est (c.à.d. avec un angle de direction de 0).

Commande de rotation du vaisseau pour le joueur (15 minutes)

Pour que le joueur puisse tourner le vaisseau, 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 votre code de gestion des *frames* de votre boucle d'animation.) Si une des touches de changement de direction est appuyée, modifiez la direction du vaisseau via l'appel d'une méthode. Cette méthode devra recevoir en paramètre le Δt et modifiera la direction du vaisseau (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*.

Méthode *move* et collision sur les bords (15 minutes)

Rajoutez une méthode *move* dans votre classe *Spaceship*. Celle-ci prendra en paramètre le Δt ainsi que le contexte graphique du *canvas*. Calculez la distance horizontale et verticale grâce au cosinus et sinus de l'angle de direction(*direction*), de la vitesse (*speed*) et du Δt, et modifiez les propriétés *x* et *y* en conséquence. Vous pouvez déjà tester que tout fonctionne en modifiant la vitesse par défaut du vaisseau dans votre constructeur (mais n'oubliez pas de la remettre à 0 après vos tests).

Modifiez la méthode *move* pour que le vaisseau ne puisse pas sortir du *canvas*. Le code est proche de ce qui a été fait au TP de la "*balle qui rebondit sur les bords*" mais en ne provoquant cette fois pas de rebond. Il vous faut juste vous assurez que les propriétés *x* et *y* restent bien à l'intérieur des dimensions du *canvas*.

Commande d'accélération et de décélération du vaisseau (15 minutes)

De la même manière que vous avez géré les touches A et D, gérez les touches W et S pour réciproquement accélérer ou freiner le vaisseau. Si une des touches de changement de vitesse est appuyée, modifiez la vitesse du vaisseau via l'appel d'une méthode. Cette méthode devra recevoir en paramètre le Δt et modifiera la vitesse (*speed*) en y ajoutant (si on accélère) ou soustrayant (si on freine) une valeur calculée en fonction du Δt et de l'accélération du vaisseau (*acceleration*).

Faites que la vitesse du vaisseau ne puisse jamais être négative, et aussi qu'elle ne puisse jamais dépasser la vitesse maximum (propriété *maxSpeed*).

Force de frottement (10 minutes)

Rajoutez une méthode de frottement dans votre classe *Spaceship*, celle-ci recevra un Δt et devra simplement réduire la vitesse d'une valeur calculée avec le Δt et coéf. de frottement (propriété *friction*) . N'oubliez pas que la vitesse du vaisseau ne peut pas être négative. Enfin, rajouter un appel à cette méthode dans votre boucle d'animation.

Génération des cibles: première version (20 minutes)

Toutes les secondes, une nouvelle cible doit apparaître, Celles-ci sont simplement des cercles et vous pouvez donc utiliser la classe *Circle* déjà existante dans votre projet. Codez un *timer* qui ajoutera un nouveau cercle d'un rayon de 20px et de position et couleur aléatoire dans une liste de cercles (c.à.d. les cibles). Dessinez les cercles dans votre boucle d'animation. Si votre tableau contient 10 cercles après l'ajout d'un cercle, le jeu est fini (le joueur a perdu). Il vous faut donc stopper la boucle d'animation et écrire le texte "You lose !" au milieu du *canvas*. **Remarque**: ne pas effacer le *canvas* après avoir affiché le texte (sinon il ne sera jamais visible puisque la boucle d'animation ne le redessinera pas).

Génération des cibles: deuxième version (20 minutes)

Le risque avec cette méthode de génération aléatoire, c'est qu'un cercle apparaisse juste à côté (ou même sur) le vaisseau du joueur (ce qui rendrait le jeu trop facile). Vous allez donc ne pas accepter les cercles générés à une position se trouvant à moins de 200px du vaisseau. Comme vous aurez besoin par la suite de tester les collisions entre les cercles et le vaisseau, mieux vaut faire d'une pierre deux coups en ajoutant une méthode *isCollidingWith* dans la classe *Circle*. Cette méthode prendra en paramètre une position x, y et un rayon (c.à.d. la représentation d'un cercle), et retournera *true* si le cercle actuel est en collision avec (ou plutôt **intersecte**) le cercle fournit en paramètre. Pour ce faire, utilisez la formule trigonométrique du cercle. Voilà le code permettant de savoir si un cercle en position (*x1*, *y1*) de rayon *r1* est en collision avec un cercle en position (*x2*, *y2*) de rayon *r2* :

```
(x2-x1)^2 + (y2-y1)^2 <= (r2+r1)^2
```

Une fois cette méthode ajoutée, utilisez la pour votre génération des cercles. L'algorithme deviendra donc: générer un cercle à une position aléatoire, si ce cercle est en collision (intersection) avec un cercle fictif de rayon 180 entourant le vaisseau, recommencer la génération aléatoire, sinon ajouter le cercle à la liste des cibles. Le reste du code reste le même (pour ce qui est du test de la fin du jeu).

Gestion des collisions entre le vaisseau et les cibles (15 minutes)

Il faut supprimer les cibles que le vaisseau touche. Dans votre boucle d'animation, rajoutez donc un code permettant de supprimer du tableau des cibles, tous les cercles actuellement en collision avec le vaisseau. L'utilisation de la méthode *filter* ainsi votre méthode de test de collision (*isCollindingWith*) devrait permettre de le faire en quelques lignes seulement. Pour une détection de collision plausible, testez avec un cercle fictif autour du vaisseau possédant un rayon de 10px.

Affichage du score (10 minutes)

Finalement, rajoutez l'affichage du score du joueur. Le score est simplement l'affichage du temps qui s'est écoulé depuis le début de votre boucle d'animation arrondie au dixième de seconde. Affichez le dans votre boucle d'animation, via la méthode *fillText* en haut à gauche du *canvas*.

Débuggage, relecture, zip, et envoi de l'email: réserve de 30 minutes