COLLÈGE DE
BOIS-DE-BOULOGNE

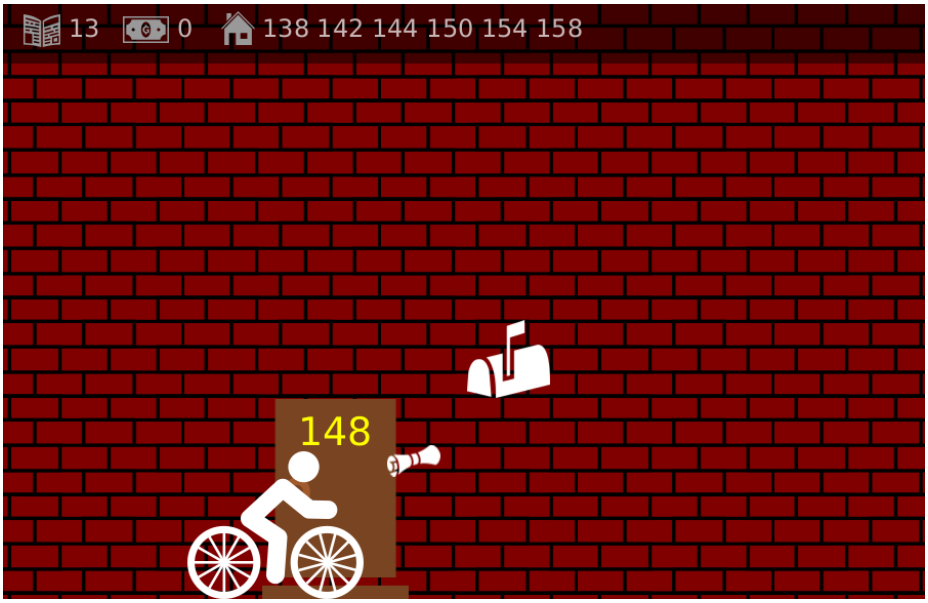
TP2 – *Camelot à vélo*

SIM – A25 – Développement d'applications dans un environnement graphique

Nicolas Hurtubise et Raouf Babari

Contexte

Le TP final consiste à programmer un jeu en interface graphique, toujours avec la bibliothèque graphique *JavaFX*.



Vous incarnez un camelot qui doit livrer des journaux à vélo dans les boîtes aux lettres des maisons abonnées.

On contrôle le camelot au clavier :

- `→` pour aller plus vite
- `←` pour aller moins vite
- `Espace` ou `↑` pour sauter
- `Z` pour lancer un journal vers le haut
- `X` pour lancer un journal vers l'avant
- Maintenir `Shift` enfoncé pour lancer les journaux avec plus de force

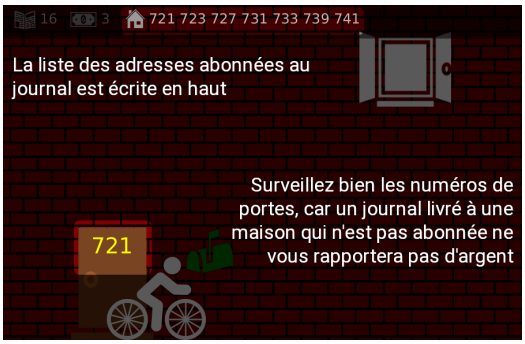
Le camelot ne peut jamais s'arrêter. La partie se termine quand il n'y a plus de journaux.

Règles du jeu

Le jeu se joue un niveau à la fois. Au début de chaque niveau, le camelot obtient des nouveaux journaux à aller distribuer aux maisons abonnées de la rue. Chaque niveau comporte 12 maisons, qui ont chacune une porte avec l'adresse affichée, une boîte aux lettres, et entre 0 et 2 fenêtres.

Seules certaines maisons sont abonnées au journal. La liste des adresses abonnées est affichée dans la barre du haut.

Pour livrer ses journaux, le camelot les lance depuis son vélo. Pour chaque journal qui tombe dans une boîte aux lettres d'une maison abonnée, le camelot gagne 1$. Si le camelot lance un journal dans la boîte aux lettres d'une maison qui n'est pas abonnée, il ne gagne pas d'argent. Un deuxième journal dans la même boîte aux lettres n'a pas d'effet.



Les maisons ont des fenêtres, et lancer un journal dans une fenêtre va la casser. Casser la fenêtre d'une maison abonnée va *coûter* 2$ au camelot en frais de dommages. À l'inverse, casser la fenêtre d'une maison non-abonnée sera récompensé par le patron du camelot à l'éthique douteuse, qui versera 2$ en récompense.

Le nombre de journaux restants et la quantité d'argent accumulée sont affichés dans la barre du haut. **Notez que la quantité d'argent peut devenir négative**, si le camelot brise des fenêtres et que ça lui coûte plus cher que son salaire.

Le jeu s'arrête lorsque le camelot n'a plus de journaux à lancer et qu'il n'en reste pas sur l'écran (on veut laisser la chance au dernier journal lancé de possiblement tomber dans une boîte aux lettres avant de déclarer que le jeu est fini).

À partir du deuxième niveau et pour tous les niveaux suivants, le camelot doit livrer l'*Édition Électronique* du Journal. L'édition électronique est affectée par des forces électriques générées par des particules chargées placées au hasard dans le niveau.

Niveau

Le jeu commence avec un écran de chargement de niveau, qui affiche en vert sur fond noir :

Cet écran dure 3s avant que le jeu ne commence, puis on commence le premier niveau. Quand on commence un niveau, le camelot reçoit +12 journaux dans son inventaire. Les journaux qui ne sont pas lancés pendant un niveau sont conservés lors du prochain niveau.

La disposition du niveau est générée au hasard au début du niveau : 12 maisons sont créées. Les adresses des maisons sont définies à partir de l'adresse de la première maison, choisie au hasard entre `100` et `950`. Chaque nouvelle maison a l'adresse `+2` par rapport à celle d'avant.

Chaque maison a une chance sur deux d'être abonnée au journal.

Les maisons sont positionnées aux coordonnées `x=1300, 2600, 3900, ...` jusqu'à la dernière du niveau qui est à la coordonnée `x=15600`. Ces coordonnées représentent *la position gauche* de la porte. La boîte aux lettres et les fenêtres de la maison sont positionnées par rapport à celles-ci.

Chaque maison possède une **boîte aux lettres**, positionnée à 200px à droite de la coordonnée `x` de la maison, et à un `y` choisi au hasard entre 20% et 70% de la hauteur de l'écran.

Une maison a un nombre aléatoire de fenêtres, choisi parmi 0, 1 ou 2. En `y`, les fenêtres sont placées à 50px du haut de l'écran. En `x`, la première fenêtre est placée à 300px à droite de la coordonnée `x` de la maison. La deuxième fenêtre (s'il y en a une) est à 600px de la coordonnée `x` de la maison.

À partir du deuxième niveau, on ajoute également des *particules chargées*, positionnées au hasard dans toute la largeur et toute la hauteur du niveau. Le nombre de particules est déterminé par le numéro du niveau actuel : $N_{\text{particules}} = min((\text{niveau} - 1) * 30, 400)$ [1]

Le niveau est considéré comme complété à partir du moment où le camelot dépasse la coordonnée `x` de la dernière maison, plus 1.5x la largeur de l'écran.

Lorsqu'on termine un niveau, on affiche l'écran de chargement du prochain niveau (ex.: ***Niveau 2*** en vert sur un fond noir, encore pendant 3s), puis on recommence avec le prochain niveau : le camelot est replacé au début, on ajoute +12 journaux à son inventaire, on recrée des nouvelles maisons et des nouvelles particules, etc.

Quand le camelot manque de journaux et qu'il n'en reste plus sur l'écran, le jeu se termine. On affiche le message de fin en rouge et en vert, avec le total d'argent collecté pendant la partie. Au bout des 3s, on recommence le jeu à partir du niveau 1.

[1] Remarquez ici que la formule donne 0 pour le niveau 1. Le premier niveau n'est pas un cas spécial

Le Camelot

Le camelot est affiché sur l'écran en alternant entre deux images : `camelot1.png` et `camelot2.png`. L'animation est créée en changeant d'image à toutes les 0.25 secondes. Utilisez la formule suivante pour déterminer l'image à afficher en fonction du *temps total écoulé depuis le début* :

$$\text{index} = \lfloor \text{tempsTotal} * 4 \rfloor \% 2$$

où $\lfloor x \rfloor$ est la fonction *plancher* (ie, arrondir vers le bas).

Le contrôle du camelot se fait avec les flèches du clavier gauche-haut-droite et la barre espace.

De base, si on ne touche à rien, le camelot va à une vitesse de $400px/s$ vers la droite. On peut cependant le faire aller plus vite ou moins vite en utilisant les flèches gauche et droite.

Si la touche gauche est enfoncée, on donne une *accélération* vers la gauche de $300px/s^2$, ce qui va faire diminuer graduellement la vitesse du vélo. Pour éviter de le faire s'arrêter ou reculer, on doit limiter la vitesse en x à être au minimum $200px/s$ vers la droite.

Si la touche droite est enfoncée, on donne une *accélération* vers la droite de $300px/s^2$. On va limiter cette vitesse à maximum $600px/s$.

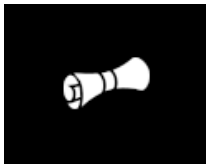
Dans le cas où aucune touche n'est enfoncée, le vélo devrait graduellement revenir à sa vitesse de base de $400px/s$. On doit donner une accélération de $300px/s^2$ vers la droite ou vers la gauche, choisi automatiquement pour que la vitesse en x du vélo retourne graduellement à sa vitesse de base de $400px/s$.

Si on appuie sur Espace ou sur la flèche du haut, le camelot saute : on devrait changer la vélocité en Y pour la mettre instantanément à $500px/s$ vers le haut. Notez que le camelot ne peut pas sauter s'il se trouve déjà dans les airs.

La gravité qui s'applique à tous les objets (dont le Camelot) donne une accélération $a_g = 1500px/s^2$ vers le bas.

L'image du camelot est un rectangle de largeur 172px et de hauteur 144px.

Journaux



Un journal est affiché avec l'image `journal.png`. Au moment de vérifier les collisions, un journal est considéré comme un rectangle de largeur 52px et de hauteur 31px.

Les journaux subissent la même gravité que le camelot, une accélération $a_g = 1500 px/s^2$ vers le bas.

Les journaux ont une *masse*. Au début du niveau, on choisit au hasard la masse que les journaux vont avoir. La masse est un nombre aléatoire entre 1kg et 2kg (ex.: 1.2281kg). La masse des journaux reste la même tant qu'on est dans un même niveau.

On peut lancer un journal en appuyant sur `Z` ou sur `X`. Si on maintient Shift en même temps, le journal est lancé *plus fort*. Le journal est créé au centre du camelot, à la même vélocité que le camelot, puis il se fait lancer avec une *impulsion* : on lui rajoute une *quantité de mouvement* initiale.

Rappel : la quantité de mouvement est simplement la masse multipliée par la vélocité :

$$\vec{p} = m\vec{v}$$

Si on connaît la quantité de mouvement initiale qu'on souhaite donner, on peut simplement utiliser :

$$\vec{v}_{\text{initiale}} = \vec{v}_{\text{camelot}} + \frac{\vec{p}_{\text{initiale}}}{m}$$

La quantité de mouvement initiale à utiliser dépend de si on a appuyé sur `Z`, `X` et `Shift` :

- `Z` = Lancer en haut : quantité de mouvement initiale de $(900, -900)$
- `X` = Lancer en avant : quantité de mouvement initiale de $(150, -1100)$

Si on maintient `Shift` pendant qu'on appuie sur `Z` ou `X`, on devrait **multiplier ce vecteur par x1.5** pour donner plus de vitesse au journal.

Le camelot ne peut pas lancer tous ses journaux d'un seul coup : on doit attendre 0.5s après avoir lancé un journal avant de pouvoir en relancer un autre.

On doit supprimer les journaux du jeu éventuellement, mais faites attention : on devrait seulement supprimer les journaux qu'on ne voit plus et qui ne vont probablement pas redevenir visibles sur l'écran à court terme.

Un journal **peut** sortir de l'écran par en haut et retomber dans l'écran plus loin. Un journal qui sort de l'écran par la gauche, par la droite ou par le bas et qui n'est plus du tout visible doit être supprimé du jeu.

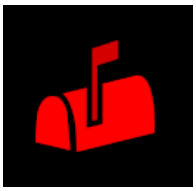
Finalement, le module de la vélocité (en anglais: la `magnitude()`) ne doit jamais dépasser 1500px/s. Si le module commence à être plus grand que ça, utilisez :

```
double max = 1500;
velocite = velocite.multiply(max / velocite.magnitude());
```

Ce bout de code sera particulièrement important quand vous ajouterez les particules chargées au code.

Boîtes aux lettres





Les boîtes aux lettres sont affichées avec les images `boite-aux-lettres.png` fournies. Au niveau des collisions, elles sont représentées par des rectangles de largeur 81px et de hauteur 76px.

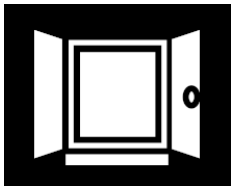
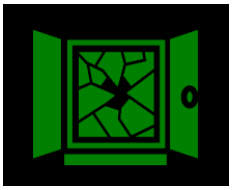
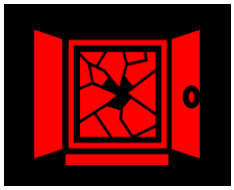
Quand un journal entre en collision avec une boîte aux lettres, il disparaît du jeu.

S'il s'agissait de la boîte aux lettres d'une maison abonnée, la boîte aux lettres change de couleur pour le vert (image: `boite-aux-lettres-vert.png`) et le camelot gagne 1$.

S'il s'agissait de la boîte aux lettres d'une maison qui n'était pas abonnée, la boîte aux lettres devient plutôt rouge (image: `boite-aux-lettres-rouge.png`) et le camelot ne gagne pas d'argent.

Si un journal entre en collision avec une boîte aux lettres déjà touchée par un autre journal, le journal disparaît, sans faire gagner d'argent.

Fenêtres

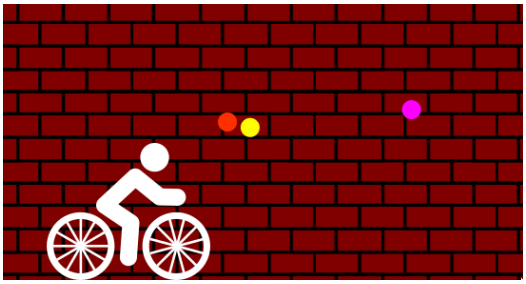




Les fenêtres sont affichées avec les images `fenetre.png` fournies. Elles sont représentées par des rectangles de largeur 159px et de hauteur 130px.

Comme pour les boîtes aux lettres, un journal disparaît du jeu lorsqu'il entre en collision avec une fenêtre. La fenêtre se brise alors et change de couleur (`fenetre-brisee-vert.png` ou `fenetre-brisee-rouge.png`). Si on casse la fenêtre d'une maison abonnée, la fenêtre devient rouge et le camelot perd 2$. Si on casse la fenêtre d'une maison qui n'était pas abonnée, elle devient plutôt verte et le camelot gagne 2$.

Lancer un journal dans une fenêtre déjà cassée fait disparaître le journal mais n'a pas d'impact sur l'argent.

Particules chargées



À partir du deuxième niveau, on voit apparaître des *particules chargées électriquement*. Les particules sont dessinées avec des petits cercles dont le *rayon* est de 10px (ne vous mélangez pas, le diamètre est de 20px).

La couleur des particules chargées est choisie au hasard. Pour avoir des couleurs vives, choisissez une couleur au hasard avec :

```
double teinte = // random entre 0 et 360
couleur = Color.hsb(teinte, 1, 1);
```

`teinte` est un nombre `double` initialisé au hasard entre 0 et 360, et correspond à un angle en degrés dans la roue des couleurs. [2]

Ces particules génèrent un champ électrique dans tout le niveau. Elles sont placées à des positions fixes (elles n'ont pas de vélocité ni d'accélération) et elles ne subissent aucune collision.

Chaque particule a une charge de $q = +900C$. On choisira en guise de *constante de Coulomb* une valeur arbitraire de $K = 90$ ici (ça ne correspond à rien de réaliste, mais ça donne un comportement de jeu agréable).

Le champ électrique total à une certaine position $\vec{x}_{\text{point}} = (x, y)$ peut être évalué en additionnant le champ électrique généré en ce point par chaque particule.

Le module du champ électrique en un certain point par la $i^{ème}$ particule est donné par :

$$E_i = \frac{k|q_i|}{r^2} = \frac{k|q_i|}{\text{distance}(\vec{x}_{i^{\text{ème}}\ \text{particule}}, \vec{x}_{\text{point}})^2}$$

La direction $\vec{d_i}$ de ce champ électrique correspond au *vecteur unitaire* qui part de la particule et qui va vers le point :

$$\vec{d_i} = \frac{\vec{x}_{\text{point}} - \vec{x}_{i^{\text{ème}}\ \text{particule}}}{||\vec{x}_{\text{point}} - \vec{x}_{i^{\text{ème}}\ \text{particule}}||}$$

(Truc : les `Point2D` ont la fonction `.normalized()` qui retourne un vecteur unitaire dans la même direction que le vecteur initial)

Important : quand la distance entre le point et la particule est trop petite, la force du champ électrique tend vers l'infini, ce qui peut nous causer des bogues. Si la distance entre le point et la particule est plus petite que 1px, on considérera une distance de 1px quand même.

[2] Voir cet article si le fonctionnement vous intéresse : https://www.learnui.design/blog/the-hsb-color-system-practicioners-primer.html

Le vecteur du champ électrique en 2D correspond donc à :

$$\vec{E_i} = E_i \vec{d_i}$$

Le champ électrique total en un point du monde correspond à la somme de tous ces vecteurs :

$$\vec{E}_{\text{total}} = \sum_{i=0}^{N-1} \vec{E_i}$$

(Si la formule vous fait peur, rappelez-vous qu'une $\sum$ n'est qu'une boucle `for()` qui additionne)

Vous devriez avoir une méthode quelque part dans votre code pour faire ce calcul. Je m'attends à trouver dans votre code quelque chose dans le style de :

```
public Point2D champElectrique(ArrayList<ParticuleChargee> particules, Point2D position)
```

Ce champ électrique va affecter les journaux lancés par le camelot. Chaque journal a une charge électrique de $q_{\text{journal}} = +900C$. Puisque la charge des particules est de même signe que celle des journaux, les journaux seront repoussés par les particules.

Pour éviter de rendre les choses trop complexes, va faire une simplification : **les journaux ne contribuent pas au champ électrique**, et n'ont donc pas d'influence les uns sur les autres.

Le champ électrique va appliquer une **force** qui va s'ajouter à la gravité qui s'applique sur les journaux. La force électrique résultante sera :

$$\vec{F}_{\text{électrique}} = \vec{E}_{\text{total}} * q_{\text{journal}}$$

Cette force va résulter en une accélération :

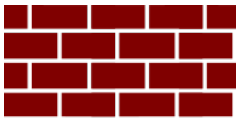
$$\vec{F}_{\text{électrique}} = m_{\text{journal}} \vec{a}_{\text{champ électrique}}$$
$$\implies \vec{a}_{\text{champ électrique}} = \frac{\vec{F}_{\text{électrique}}}{m_{\text{journal}}}$$

L'accélération totale à chaque `update()` sera : $\vec{a} = \vec{a}_{\text{gravité}} + \vec{a}_{\text{champ électrique}}$

Pour résumer : pour chaque journal, à chaque `update()` du jeu, on va :

1. **Calculer le champ électrique qui s'applique au centre du journal exercée**
   - On fait la somme des champs électriques exercés en ce point pour toutes les particules
2. **Mettre à jour l'accélération** du journal
   - Calculer la force électrique
   - Trouver l'accélération dûe à la force électrique
   - L'accélération causée par la gravité doit également être considérée
   - L'accélération totale sera donc $\vec{a} = \vec{a}_{\text{gravité}} + \vec{a}_{\text{champ électrique}}$
3. **Mettre à jour la vélocité** selon l'accélération calculée à partir des forces, **puis de la position** selon la vélocité
   - Similaire à ce qu'on fait avec les autres objets qui bougent sur le jeu, inspirez-vous du code vu en classe

Décor



L'arrière-plan du jeu doit être noir et être recouvert de briques (image: `brique.png`).

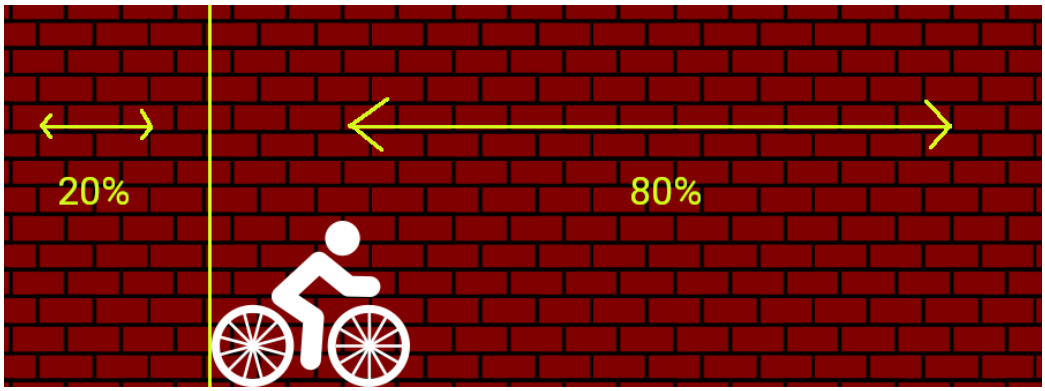
Chaque brique a une largeur de 192px et une hauteur de 96px.

Caméra

La caméra doit suivre le Camelot à mesure qu'il se déplace dans le niveau (*à lire là-dessus : notes de cours `JavaFX - Animations partie 5`*).

La caméra doit toujours pointer le monde à partir de 20% de la coordonnée gauche du camelot.

En d'autres termes, sa position devrait toujours être égale à la position x du camelot moins 20% de la largeur de l'écran.



Débogage

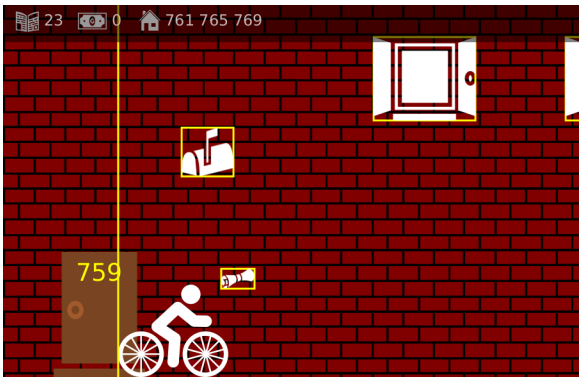
> ➢ N'attendez **PAS** la toute fin du TP pour coder les choses demandées dans cette section. Faites-le tôt dans votre développement.
>
> Ce mode va vous aider à déboguer votre code si les collisions ne marchent pas ou si des choses n'apparaissent pas correctement à l'écran.

À tout moment pendant le jeu, on devrait pouvoir **appuyer sur D pour activer/désactiver l'affichage de débogage**.

Lorsque ce mode est actif, le contour des rectangles de collisions doit s'afficher en jaune (utilisez `context.strokeRect()`).

Ce rectangle doit au moins s'afficher autour des boîtes aux lettres, journaux et fenêtres. Si vous voulez afficher un rectangle jaune autour des autres objets (par exemple, si vous trouvez ça plus facile de le coder une seule fois pour tous les objets), vous pouvez le faire.

Pour vérifier que la caméra regarde la bonne chose, on doit également dessiner une ligne verticale jaune à 20% de l'écran (*ie*, là où l'image de vélo devrait commencer si la caméra fait bien son travail).



Débogage de la logique du jeu

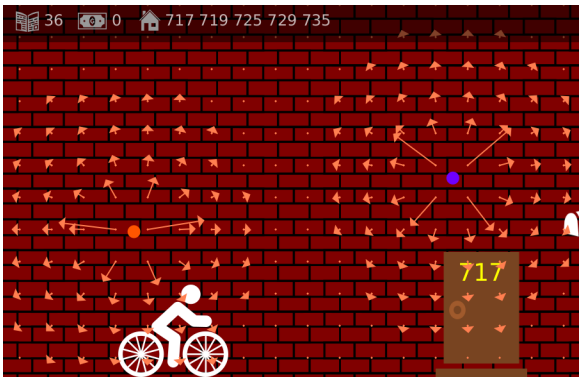
À tout moment, on peut appuyer sur une des touches suivantes pour tester des choses spécifiques :

- Appuyer sur la touche `Q` doit ajouter +10 journaux au camelot
- Appuyer sur la touche `K` doit mettre le nombre de journaux à zéro (ce qui devrait avoir pour effet de mettre fin à la partie)
- Appuyer sur la touche `L` doit faire passer au prochain niveau (on devrait alors voir l'écran de chargement du prochain niveau)

Débogage du champ électrique

Pour vous assurer que votre champ électrique fonctionne, on doit activer/désactiver une *visualisation du champ électrique* avec des flèches lorsqu'on appuie sur `F`.

La visualisation du champ devrait se faire en calculant le vecteur de champ à tous les 50px en x et en y. Une méthode qui dessine un vecteur (ie, un `Point2D`) sur l'écran vous est fournie dans la classe `UtilitairesDessins`.

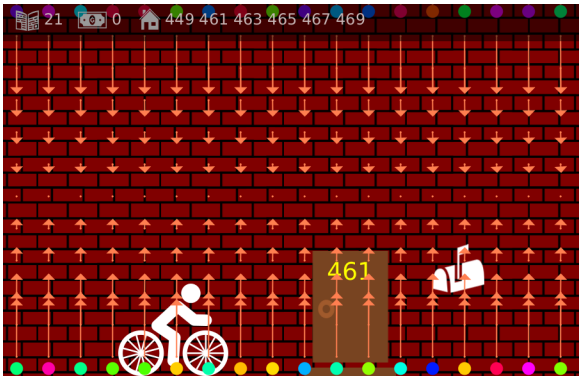


Voici une esquisse de code pour vous aider :

```
for (double x = 0; x < LARGEUR_NIVEAU; x += 50) {
    for (double y = 0; y < HAUTEUR_ECAN; y += 50) {

        var positionMonde = new Point2D(x, y);
        var positionEcran = // calculez ça selon votre objet Camera

        // TODO: Seulement faire ça si la position (x, y) est visible dans l'écran

        Point2D force = champElectrique(particules, positionMonde);

        UtilitairesDessins.dessinerVecteurForce(
                positionEcran,
                force,
                contexteGraphique
        );
    }
}
```

Lorsqu'on appuie sur la touche `I`, on doit mettre des particules de test sur l'écran. On supprime toutes les particules déjà présentes dans le niveau et on ajoute plutôt deux lignes de particules : une ligne de particules placées à 10px du haut de l'écran, espacées de 50px horizontalement, et une deuxième ligne similaire à 10px du bas de l'écran (également espacées de 50px horizontalement).

Ça devrait donner le résultat suivant :



Interface

La fenêtre du programme doit avoir une largeur de $900px$ et une hauteur de $580px$.

- La fenêtre ne doit **pas** être redimensionnable (*resizable*) : la taille de la fenêtre est fixée au début et elle ne peut pas être redimensionnée avec la souris
- La fenêtre doit avoir en guise de petit logo en haut à gauche l'image `journal.png` fournie
- La fenêtre doit porter le titre `"Camelot à vélo"`

La scène de jeu contient *un seul* `Canvas` (l'écran de jeu), qui doit occuper toute la fenêtre.

=> Si on appuie sur Escape, le jeu se ferme

Code & Design Orienté Objet

Ce sera à vous de choisir le découpage en classes optimal pour votre programme. Faites bon usage de l'orienté objet et de l'héritage lorsque nécessaire.

Vous serez évalués sur la qualité de votre code. Vous serez en particulier pénalisés pour les copiers-collers de code, trouvez des façons pour les éviter.

Comme dans le TP1, votre code qui définit les composantes JavaFX (le `Canvas`, l'`AnimationTimer`, etc) ne devrait **pas** être responsable de faire de la logique de jeu. Assurez-vous de concevoir vos classes en gardant ça en tête.

Inspirez-vous des derniers exemples qu'on a vus dans les notes de cours (à relire: `JavaFX - Animations partie 5`).

Éléments fournis

Mis à part la classe `UtilitairesDessins`, aucun code n'est fourni pour démarrer le projet, mais vous pouvez (et vous devriez) vous inspirer des exemples de code vus en classe lors des différents chapitres sur les *Animations en JavaFX*.

Les images nécessaires au programme sont fournies et sont basées sur différentes images provenant du site https://game-icons.net/ (presque toutes les images) et du site https://openclipart.org (le vélo).

Si vous préférez utiliser d'autres images à la place, vous pouvez le faire `:-)`

Conseils pour bien commencer

1. Travaillez ensemble, surtout au début

Certaines parties du TP se séparent bien pour que les deux membres de l'équipe travaillent chacun de leur côté, d'autres, moins bien.

Au début du projet, on vous recommande de travailler en **programmation par binômes**, à deux sur le même ordinateur.

Si ça se passe très très bien, vous pouvez essayer de vous séparer les tâches (en gardant ça équitable entre les membres), mais sentez-vous libres de continuer le travail en binôme tout le long si vous préférez ça.

2. Allez-y par étapes

N'essayez **pas** de tout coder d'un coup avant de tester, ça ne marchera pas, il y a trop de choses.

Commencez par faire marcher une première étape relativement simple, par exemple : mettre le Camelot à vélo sur l'écran, qui ne bouge pas, qui n'est pas animé, qui ne peut pas lancer de journaux.

Une fois que ça marche et que c'est bien testé, ajoutez un autre petit truc, par exemple : le déplacement du Camelot à une vitesse fixe (qui va seulement aller vers la droite et sortir de l'écran tant qu'il n'y a pas de caméra).

Une fois que ça marche et que c'est bien testé, ajoutez encore un autre petit truc, par exemple : pouvoir sauter et retomber au sol.

Une fois que ça marche et que c'est bien testé, ajoutez encore autre chose, par exemple : pouvoir lancer un journal, sans force électrique.

etc

Allez-y avec une série de petites étapes, et vous verrez qu'un jeu relativement complexe n'est en réalité rien d'autre qu'un gros tas de petites étapes relativement simples.

Ne passez pas à la prochaine étape avant d'avoir bien testé celle d'avant.

Enregistrement des équipes

➢ Le travail est à faire en **équipes de deux**.

Remplissez le formulaire ici pour enregistrer votre équipe :

https://forms.cloud.microsoft/r/MjuSUY2BJJ

Vous DEVEZ être en équipe avec quelqu'un du même groupe que vous.

Remise

Faites une seule remise par équipe sur Léa.

Vous devez remettre sur Léa votre projet IntelliJ (avec la configuration Gradle, le code, les ressources, etc.) **dans un fichier .zip**. La date de remise est spécifiée sur Léa.

Barème

- 70% : Fonctionnalités demandées implantées correctement
  - (5%) Touches de débogage de l'application
  - (10%) Physique/contrôle/dessin du camelot
  - (10%) Génération des adresses abonnées et affichage des maisons
  - (10%) Physique des journaux (lancer, gestion correcte excluant l'électricité)
  - (6%) Affichage des infos sur l'écran (journaux restants, argent, maisons abonnées)
  - (8%) Fenêtres, boîtes aux lettres et collisions avec les journaux
  - (3%) Affichage de l'arrière-plan en briques
  - (6%) Logique de jeu générale (génération du niveau, écran de chargement, fin de partie, etc)
  - (7%) Particules chargées et application de l'accélération sur les journaux
  - (5%) Gestion correcte de la Caméra
- 30% : Qualité du code
  - (5%) Séparation correcte entre la logique du jeu et la déclaration d'interface JavaFX
    - Logique du jeu (ex.: classe `Partie`) bien isolée de l'`AnimationTimer`
  - (5%) Utilisation judicieuse de l'héritage
  - (20%) Qualité générale du code
    - Code bien commenté lorsque nécessaire
    - Respect des conventions : minusculeCamelCase pour les variables/méthodes, MajusculeCamelCase pour les noms de classes, noms de variables/méthodes clairs, respect de l'indentation, etc
    - Bon découpage en méthodes, **Pas de variables globales !**
    - Encapsulation : attributs `private` (ou `protected`), avec getters/setters au besoin
    - **Pas de copier-coller de code!** Quand quelque chose est répété, trouvez une façon de ne pas le copier-coller

La qualité du code et le fonctionnement sont deux critères différents. Toutefois, si vous n'avez pas tenté de faire l'ensemble du travail, vous ne pouvez pas avoir l'ensemble de vos points pour la qualité du code. Si des parties du travail sont absentes, vous serez pénalisé dans la qualité de la programmation en proportion de la partie du travail qui n'a pas été faite.

Note sur le plagiat

Le travail est à faire **en équipes de 2**. **Tout le monde dans l'équipe doit participer.** Si vous avez un problème avec un coéquipier ou une coéquipière qui ne travaille pas, avertissez rapidement votre professeur pour trouver une solution.

Ne *partagez pas de code avec une autre équipe que la vôtre*, même pas "juste pour aider un ami". Ça serait un **plagiat**, et **tous les membres des équipes concernées auraient la note de *zéro***.

Si jamais vous utilisez *chatGPT*, vous **devez** me citer quels bouts de code ont été générés avec quels prompts. Mieux vaut me donner plus de détails que pas assez.