Un personnage à déplacer

> À la séance précédente, nous avons affiché des images, utilisé des événements, et modifié l'état d'un objet de jeu (le coffre). Nous allons maintenant voir comment créer une classe pour gérer des objets complexes (avec une positions, un état, etc.) de manière efficace.

1. Créez un nouveau fichier `include/Personnage.h` et un autre fichier `src/Personnage.cpp`. Ajoutez les directives habituelles `#ifndef / #define / #endif` dans votre fichier `.h`. Écrivez une classe `Personnage`. Un personnage est défini par une `Image` et deux coordonnées entières, initialisables par le constructeur.

> **Note :** les objets de la class `Image` sont des objets légers (qui, en interne, se réfèrent à des pointeurs de la bibliothèque SDL) : utilisez des copies et non des références pour les intégrer à vos classes.

2. Supprimez le coffre (vous pouvez commenter les lignes que vous avez écrites dans le `case ESPACE_APPUYE` pour pouvoir les retrouver plus tard). Chargez l'image `assets/personnage_simple.png` et instanciez un object `Personnage` dans votre `main` aux coordonnées (0,0) : ce sera le personnage que vous contrôlerez pendant le jeu, vous pouvez donc lui donner un nom qui reflète cela (`Personnage chevalier;`, `Personnage heros;`, etc.). Ajoutez une méthode `dessiner` à votre classe `Personnage` : cette fonction ne prend aucun paramètre et utilise les attributs de la classe pour dessiner le personnage au bon endroit. Appelez cette méthode dans la boucle de jeu, et vérifiez que votre personnage s'affiche correctement.

3. **Rappel :** le moteur utilise, pour les coordonnées, une origine en haut à gauche, un axe X orienté de gauche à droite, et un axe Y orienté de haut en bas. Si je veux déplacer un personnage de 1 pixel vers la droite, comment dois-je modifier sa position ? Même chose vers la gauche ? Vers le haut ? Vers le bas ?

4. Implémentez des méthodes `allerDroite`, `allerGauche`, `allerHaut` et `allerBas` : ces fonctions doivent déplacer le personnage d'une case dans la direction demandée. **Rappel :** la grille fait 16 pixels de côté.

5. En utilisant les événements `GAUCHE_APPUYE`, `DROITE_APPUYE`, `HAUT_APPUYE` et `BAS_APPUYE`, faites en sorte que l'on puisse appeler ces différentes fonctions. Compilez et vérifiez que votre personnage bouge bien dans les bonnes directions lorsque vous appuyez sur les flèches. Vérifiez également que votre personnage reste aligné sur la grille (il ne doit pas se retrouver positionné à cheval sur deux cases).

6. Que se passe-t-il lorsque le personnage est à côté d'un bord et que l'on appuie sur la flèche pour aller vers ce bord ? Modifier vos fonctions `allerGauche` et `allerHaut` pour empêcher le personnage de sortir de la fenêtre par la gauche ou par le haut. Vous pouvez utiliser des conditions `if` en vous basant sur la hauteur et largeur de la fenêtre.

7. Modifiez vos fonctions `allerDroite` et `allerBas` pour empêcher votre personnage quitter la fenêtre par la droite ou par le bas : que faut-il prendre en compte, par rapport à la question précédente ? **Indice :** la question 4 de la séance 1 devrait vous donner une piste de ce qu'il faut prendre en compte pour gérer ces deux directions.

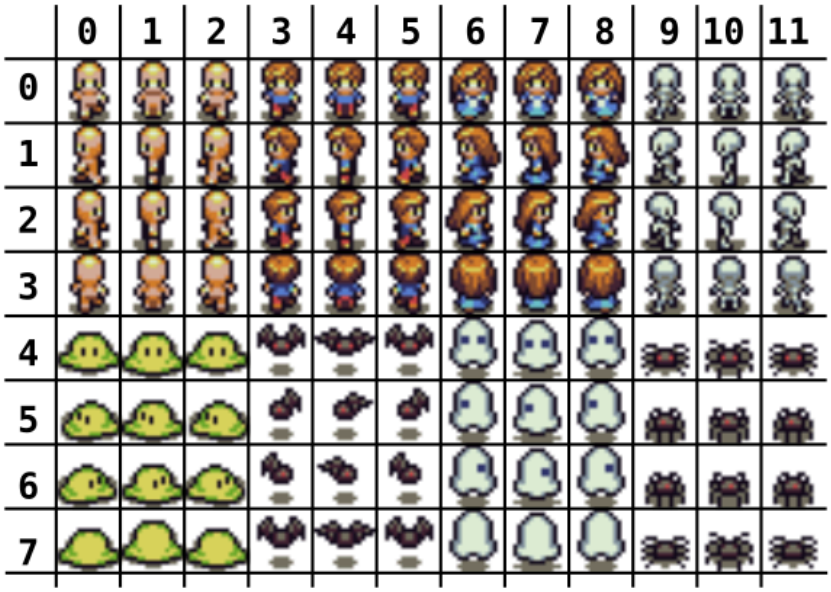
> Pour ces quatre fonctions, vous pouvez créer une sous-fonction unique `bordAtteint` qui teste si une case est au bord (il faudra ensuite simplement appeler cette fonction avec les bons paramètres — en l'occurrence, les bonnes coordonnées). Cette fonction sera très utile car on pourra la réutiliser plus tard et l'adapter à d'autres cas !

Un peu d'animation

> **Point technique :** dans un jeu, on va rarement déplacer une image fixe pour animer un personnage. Les animations sont décomposées en des images plus ou moins nombreuses pour prendre en compte les orientations possible du personnage et les différentes étapes d'une animation. Pour éviter d'avoir à charger beaucoup d'images, on utilise une feuille de *sprites* (*sprite sheet*) : un *sprite* est un morceau d'image qu'on affiche à l'écran. Au lieu d'afficher l'ensemble de l'image, on va sélectionner, à chaque affichage, une partie de l'image qu'on souhaite afficher.

8. Ouvrez le fichier `assets/personnages.png` dans votre navigateur de fichiers (pensez à zoomer !). Il s'agit d'une feuille de *sprites*. Remarquez que plusieurs personnages sont dessinés dans différentes positions, mais que pour chaque personnage, les positions sont dans le même ordre : cela permet, une fois que l'on a codé un personnage, de réutiliser le même code en décalant simplement la partie dessinée pour afficher un personnage différent.

> Numérotons les colonnes et les lignes de cette image, en partant de 0, ce qui nous permet d'indexer la feuille de *sprites* de cette façon :



Capture d'écran de la feuille de sprites, avec les colonnes et les lignes indexées.

> Ainsi, l'image du squelette qui regarde vers la droite sans bouger correspond à l'index `(10, 2)`. Les cases faisant 16 pixels de large, il s'agit de la position `(10x16, 2x16)`, soit `(160, 32)`.

9. L'image que vous utilisiez jusqu'à maintenant (`assets/personnage_simple.png`) correspond au *sprite* en colonne 4, ligne 0. Au lieu de charger `assets/personnage_simple.png`, chargez `assets/personnages.png`. Si vous compilez et lancez le programme tel quel, la feuille de *sprite* entière s'affiche (vous pouvez tester si vous le souhaitez). Dans le constructeur de la classe `Personnage`, appelez la méthode `selectionnerRectangle` pour sélectionner le *sprite* à l'index `(4,0)` (pensez à bien tout multiplier par la taille des cases). Compilez et vérifiez que votre affichage est correct : vous devriez voir la même image qu'avant votre changement d'image, avec juste votre personnage sur sa case qui regarde vers le bas.

10. On veut modifier l'apparence de notre personnage lorsqu'il bouge : on veut qu'il regarde à gauche lorsqu'il va à gauche, à droite lorsqu'il va à droite, etc. Identifier les index des sprites que vous devez afficher. **Indice :** ils sont tous sur la colonne 4.

11. Modifier vos fonctions `allerDroite`, `allerGauche`, `allerHaut` et `allerBas` : en plus de modifier les coordonnées, ces fonctions doivent sélectionner le bon rectangle dans la feuille de *sprites*. Compilez et vérifiez que le personnage se tourne du bon côté lorsque vous le déplacez.

Changer de peau

12. On veut pouvoir créer un autre type de personnage, et donc utiliser un autre *skin* (la « peau » de notre personnage) que celui du chevalier. Prenons par exemple le squelette : au lieu de sélectionner la case `(4,0)` pour l'image du personnage qui regarde en bas, il faudrait donc sélectionner la case `(10,0)`. Pour celle qui regarde vers la gauche, la case `(10,1)` au lieu de `(4,1)`. Comment pouvez-vous facilement permettre la sélection d'un autre *skin* (le squelette au lieu du chevalier par exemple) ?

13. Ajoutez deux attributs `skin_x` et `skin_y` qui correspondent à la position de votre skin. Ajoutez également les paramètres correspondant dans votre constructeur. Mettez à jour vos fonctions pour `allerDroite`, `allerGauche`, `allerHaut` et `allerBas` pour que la sélection du rectangle soit faite *relativement* à `skin_x` et `skin_y`. **Exemple :** le fantôme à son *skin* qui commence à l'index `(6,4)`. Son *sprite* qui regarde à droite est à la position `(7,6)`. Dans votre code, vous allez donc chercher la position `(_skin_x+1,skin_y+2)`, ce qui correspondra à la position « regarde à droite » pour n'importe quel *skin* de la feuille.

14. Modifiez le code dans `main` pour prendre en compte les nouveaux paramètres de votre constructeur : pour le chevalier, sélectionnez le skin avec `skin_x=3` et `skin_y=0`. Compilez, exécutez et vérifier que votre affichage fonctionne toujours lorsque vous déplacez le personnage. Essayez de modifier le *skin* de votre personnage (remplacer les coordonnées du *skin* par celles du squelette, du fantôme, etc.)., et vérifiez que vous continuez d'avoir un affichage correct lorsque vous déplacez le personnage.

Des ennemis !

15. Remettez le *skin* du chevalier à votre personnage (ou un autre si vous préférez, c'est votre personnage, c'est vous qui décidez de son apparence). Instanciez deux autres personnages, `ennemi1` et `ennemi2`, et donnez-leur des *skins* adaptés (de même, vous êtes libres : squelette, araignée…). Initialisez l'un des deux ennemis à la case `(5*TAILLE_CASE, TAILLE_CASE)` et l'autre à la case `(TAILLE_CASE, 5*TAILLE_CASE)`. Ajouter leurs fonctions d'affichage dans la boucle du jeu. Compilez et vérifiez qu'ils apparaissent aux bons endroits.

16. Ces deux nouveaux personnages sont des ennemis : ils ne sont pas contrôlés par la personne qui joue mais par l'ordinateur (on peut parler de « bots » ou d'IA – intelligence artificielle). Dans notre cas, on va faire une IA rudimentaire : les ennemis vont marcher en ligne droite et repartir en arrière lorsqu'ils arrivent au bord de la fenêtre. L'ennemi 1 ira de haut en bas et de bas en haut, l'ennemi 2 de droite à gauche et de gauche à droite. Ajouter à votre classe `Personnage` un attribut pour retenir la direction courante de votre personnage. Pensez à ajouter un paramètre à votre constructeur pour pouvoir initialiser cette direction (vous pouvez mettre une valeur par défaut à ce paramètre, par exemple faire quel que le personnage regarde vers le bas au début du jeu).

> **Conseil :** vous devez donc initialiser le rectangle sélectionné dans le constructeur selon la direction choisie. Pour éviter de dupliquer les morceaux de code déjà implémentés dans `allerGauche`, etc., vous pouvez créer une méthode `mettreAJourDirection` qui choisit le bon rectangle selon la direction. Il ne vous reste plus qu'à mettre à jour l'attribut de direction dans vos fonctions `allerGauche` et d'appeler cette fonction, que vous pouvez donc également appeler dans le constructeur.

17. Nos ennemis vont avancer automatiquement. Ajoutez une méthode `avancer` à la classe personnage qui fait avancer l'ennemi dans sa direction courante. Si un ennemi atteint le bord (pensez à réutiliser votre fonction `bordAtteint`), alors on inverse sa direction.

    Dans la boucle du jeu, avant les fonctions d'affichage, appelez cette méthode pour vos deux ennemis.

    Compilez et exécutez. Vérifier que vos ennemis vont bien des allers-retours chacun sur sa ligne (un horizontalement, l'autre verticalement). Quel est le problème ?

18. Pour éviter que nos ennemis n'aillent trop vite, on va limiter leurs déplacements. Pour cela, on va utiliser la méthode du moteur `animationsAmettreAjour` : cette fonction renvoie vrai lorsqu'il faut mettre à jour les animations, « seulement » 8 fois par seconde (alors que la boucle du jeu tourne, on le rappelle, à 60 images par seconde !). Ajouter une condition pour que les ennemis ne bougent que lorsque `animationsAmettreAjour` renvoie `true`.

19. Si on touche l'un des ennemis, c'est perdu : ajoutez une méthode `touche` à votre classe `Personnage` qui prend un autre objet `Personnage` en paramètre, et teste si le personnage cible touche le personnage paramètre (on peut simplement tester si leurs deux coordonnées sont égales). Après les déplacements, appelez cette fonction pour vérifier si le personnage contrôlé a touché l'un des deux ennemis. Si c'est le cas, mettez le booléen `quitter` à `true` : lorsque l'on touche un ennemi, c'est perdu, et le jeu s'arrête.

> **Note :** évidemment, en pratique, un jeu ne s'arrête pas brutalement dès qu'on touche un ennemi, mais c'est une toute première version.

Pour aller plus loin…

20. **(Optionnel)** Afin de bien distinguer la fin de partie « perdu » d'une fin où on quitte manuellement le jeu, ajoutez une petite pause avant de quitter lorsqu'on touche un ennemi (utilisez la méthode `attendre` du moteur).

21. **(Optionnel)** Pour ajouter un peu de dynamisme, on souhaite ajouter des animations de marche à nos personnages. Au lieu d'une image fixe, on veut alterner entre quatre images pour chaque direction. Par exemple, pour la position « regarde vers le bas », on va alterner entre les images indexées par `(3, 0)`, `(4, 0)`, `(5, 0)` et à nouveau `(4, 0)` (l'image du milieu sert deux fois). Cet enchaînement nous donne une animation simple de marche. Pour réaliser cette animation, vous devez :

    - retenir la ligne courante. Vous pouvez utiliser, par exemple, une `enum Direction { GAUCHE, DROITE, HAUT, BAS}`, ou bien des constantes ou des `strings` si vous préférez
    - retenir à quelle position de l'animation on se trouve (image numéro 0, 1, 2 ou 3)
    - à chaque boucle d'affichage, si `animationsAmettreAjour` renvoie `true`, alors mettre à jour votre animation (et donc passer à l'image suivante dans la séquence de 4 images qui forment votre boucle) : vous pouvez ajouter une méthode `mettreAjourAnimation` à votre personnage pour faire le nécessaire. Pensez à utiliser le modulo (`%`) pour boucler 0, 1, 2, 3, 0, 1, 2, 3, etc.

Implémentez et testez. **Note :** évidemment, le personnage donne l'impression de marcher sur place sans avancer. Une version encore plus avancée consisterait à n'afficher l'image d'animation qu'au moment où le personnage marche, et à le déplacer de manière continue vers la case (au lieu de le « téléporter » instantanément sur la case suivante comme on le fait actuellement), mais vous n'aurez pas le temps d'implémenter une telle version.

22. **(Optionnel)** En l'état, le jeu n'est pas bien difficile : il suffit de ne pas bouger pour ne jamais perdre. Pour rendre le jeu plus intéressant et difficile, on va utiliser quelque chose de très utile dans le jeu vidéo : l'aléatoire. L'instruction suivante vous permet de tirer un nombre aléatoire entre 0 et 3 inclus : `int nombre = rand() % 4;` Au lieu de faire avancer les ennemis en ligne droite, tirez un nombre aléatoire à chaque mise à jour et faites-les se déplacer en haut, à droite, à gauche et en bas selon le nombre tiré.

Modifié le: lundi 25 novembre 2024, 17:52

Adresse

IUT d'Orsay - 13 avenue des sciences - 91190 Gif-Sur-Yvette FRANCE
Tél. : +33 1 69 33 60 00
Accès : RER B Le Guichet

Informations

Site web

Contacts

Accéder à l'IUT

Tableaux d'affichage