Lilian BUZER

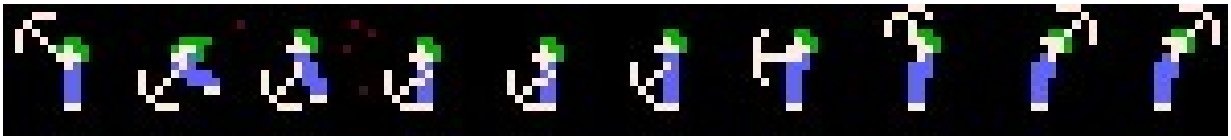
ESIEE PARIS

LEMMINGS

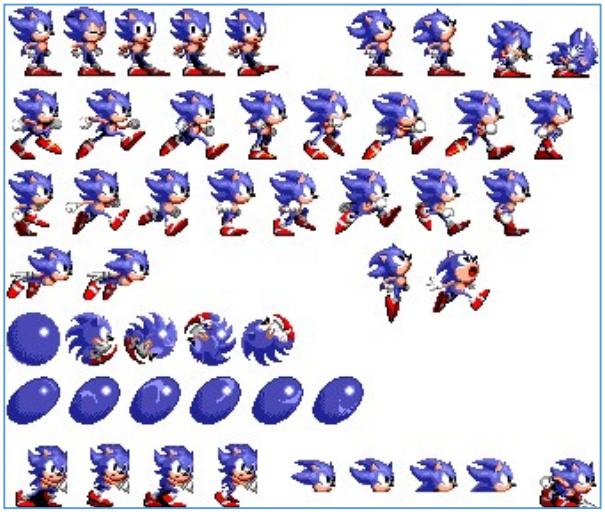
Phénomène vidéoludique de l'année 1991, Lemmings est plébiscité pour son concept singulier, à la fois simple et ingénieux mélangeant les styles de jeu de gestion et de jeu de casse-tête. Le joueur doit guider vers la sortie des dizaines de lemmings, minuscules mammifères écervelées, dans des niveaux truffés de dangers mortels en essayant qu'un maximum d'entre eux survive !

Gérer une animation

Nos lemmings effectuent une multitude d'actions. Ils creusent, ils marchent, ils grimpent… Pour produire un rendu graphique sympathique, les graphistes dessinent une animation sous forme d'une suite de sprites. Ci-dessous, nous vous présentons la série associée à l'action « creuser le sol ».



Il y a plusieurs actions et chacune comporte plusieurs sprites. Ainsi pour les lemmings, on dispose d'environ 250 sprites d'animation. Plutôt que de stocker chaque sprite dans un fichier, on préfère stocker l'ensemble dans une grande image appelée « sprite sheet » :



Sprite Sheet du jeu Sonic 1

Nous vous fournissons une planche simplifiée de sprites pour le jeu LEMMINGS. Chaque sprite est stocké dans un carré 30x30 et chaque animation occupe une ligne. Lorsqu'il n'y a plus de sprites à charger, une case rouge est présente. Voici la fonction qui charge la ligne d'animation numéro : id et qui retourne une liste de sprites :

```
LARG = 30
def ChargeSerieSprites(id):
    sprite = []
    for i in range(18):
        spr = planche_sprites.subsurface((LARG * i, LARG * id, LARG,LARG))
        test = spr.get_at((10,10))
        if ( test != (255,0,0,255) ):
            sprite.append( spr )
    return sprite
```

Nous fournissons la ligne suivante :

```
time = int( pygame.time.get_ticks() / 100 )
```

Ainsi, toutes les 0,1 secondes, la valeur de "time" augmente de 1. Nous allons nous caller sur cette valeur pour choisir le sprite à afficher dans l'animation. Si « creuse » désigne la liste de sprites de l'action « creuser », nous allons afficher :

```
creuse[time%len(tombe)]
```

En utilisant cette ligne, lorsque time progresse de 303 à 310, avec len(tombe)=4 par exemple, l'indice va varier ainsi : 3, 0, 1, 2, 3, 0, 1, 2, 3 …

Gérer des personnages

Fini la tranquillité de l'aquarium où il y avait quelques poissons à gérer, avec le jeu lemmings on passe à la vitesse supérieure. En effet dans le jeu, on peut jouer avec une centaine d'individus. On va donc utiliser une liste python dans laquelle nous allons stocker nos lemmings.

Comment représenter un personnage ? Sans utiliser l'approche programmation objet, nous pouvons utiliser un concept très proche et très plébiscité dans le monde du jeu vidéo : les dictionnaires. Ainsi un lemming est créé de la manière suivante :

```
lemming = { }
lemming['x'] = 250
lemming['y'] = 100
lemming['vx'] = -1
lemming['etat'] = ETAT.Marche
```

Pour accéder à la coordonnée en x du lemming il suffit d'écrire :

lemming['x']

On stocke ensuite ce personnage au moyen de la commande « append » :

```
lemmingsLIST  = []
lemmingsLIST.append(lemming)
```

A ce niveau, nous n'avons plus besoin de la variable lemming car pour accéder au lemming en question il suffit maintenant d'écrire :

```
lemmingsLIST [0]
```

Ainsi dans boucle principale de notre jeu, nous allons effectuer l'affichage de cette façon :

```
Afficher le fond
Pour chaque lemming dans la liste des lemmings
    lemming courant => fait ton affichage
```

Python nous fournit une syntaxe tout à fait simple pour parcourir une liste :

```
for onelemming in lemmingsLIST:
    xx = onelemming['x']
    yy = onelemming['y']
    screen.blit(…,(xx,yy))
```

A ce niveau, il faut bien saisir que onelemming est un lemming de la liste. Lorsque l'on rentre dans la boucle, il correspond au premier lemming de la liste. Ensuite, une fois l'opération blit effectuée, onelemming est mis à jour par l'instruction for et il correspond alors au 2ème lemming dans la liste et ainsi de suite jusqu'au dernier. La variable onelemming va désigner à tour de rôle chaque lemming.

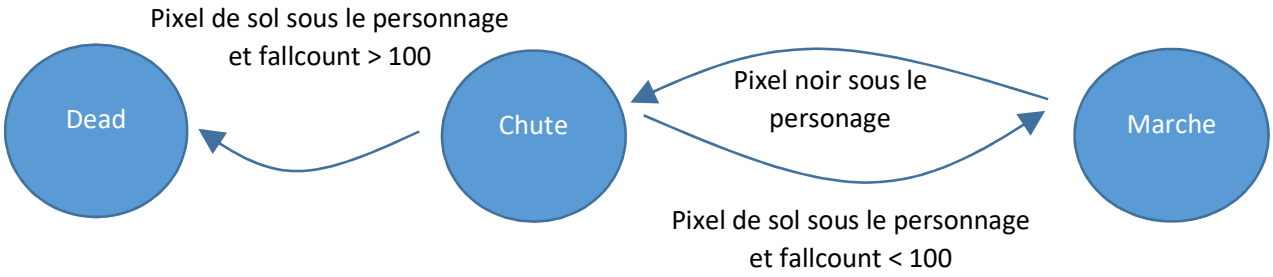
Gestion des transitions et des états

La logique pour ce jeu devient un peu plus complexe. En effet, un lemmings a une intelligence et doit se comporter différemment en fonction de l'action à effectuer. Par exemple lorsqu'il marche et qu'il se retrouve au-dessus du vide, il passe en mode « chute ». Lorsqu'il chute, on incrémente un compteur à chaque affichage pour estimer la hauteur de la chute. Lorsqu'il touche le sol si le compteur est trop important, il passe en mode « dead » sinon il recommence à marcher.

Voici comment nous allons gérer cette logique, juste après la boucle while nous allons trouver :

```
Pour chaque lemmings :
    1- Déterminez l'état courant à partir de l'état précédent et de l'environnement
    2- Appliquez l'action associée à l'état courant
    3- Affichez le lemming – le sprite dépend de l'état courant
```

Nous présentons trois états et leurs transitions : Dead, Chute, Marche. Généralement, on représente les états par des ronds et les transitions par des flèches :



Actions propres à chaque état :

- Dead : pas grand-chose à faire !
- Chute : y -= 1
- Marche : x += vx

Afin de gérer les états, chaque lemming embarque son paramètre ['etat']. Cependant, il est dangereux de représenter les états par des chiffres dans le code. En effet, pour des raisons de lisibilité, vous allez rapidement vous mélanger entre les chiffres 1,4,3,9… Un programmeur averti va préférer associer les chiffres à des noms de variable :

```
# liste des etats
EtatMarche = 100
EtatChute  = 200
EtatStop   = 300
EtatDead   = 400
```

Ainsi, au lieu d'écrire lemming['etat'] = 200 qui est peu lisible, on pourra écrire : lemming['etat'] = EtatChute ce qui est bien plus clair. Au final, la variable représente un nombre, mais la lecture du code est largement facilitée.

Pour gérer les transitions, nous pouvons utiliser la structure suivante :

```
# gestion des états et transitions
if ( etat == EtatChute ) :
    if ( condition de transition1 ) :
        etat = EtatDead
    elif ( condition de transition 2) :
        etat = EtatMarche

elif ( etat == EtatMarche ) :
        …
```

Dans un deuxième temps, nous effectuons l'action propre à l'état :

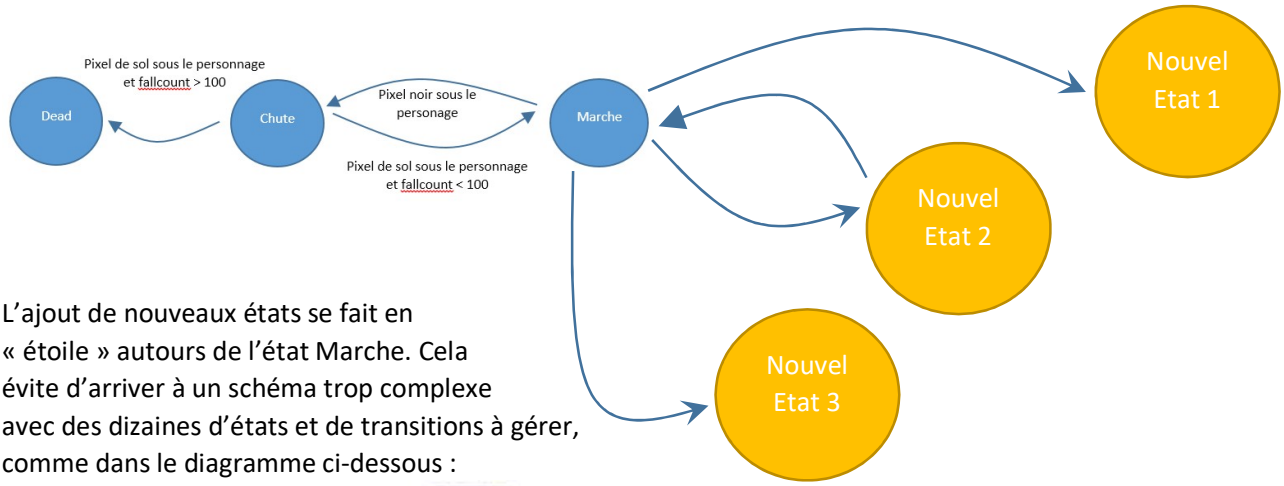
```
if ( etat == EtatChute ) :
        Actions à effectuer pour la chute
elif ( etat == EtatMarche ) :
        Actions à effectuer pour la chute
```

Dans un troisième temps, nous dessinons le sprite en choisissant l'animation associée à l'action.

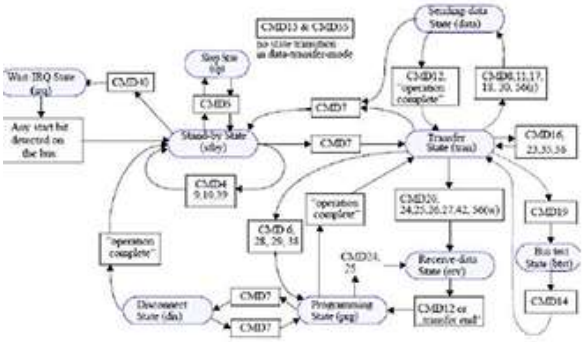
Rajouter des états

En allant plus loin dans le jeu, nous allons ajouter de nouvelles actions (creuser, bloquer…) qui vont se traduire par de nouveaux états et de nouvelles transitions. Afin d'alléger le programme et la conception du jeu, nous allons faire en sorte que les nouveaux états ne puissent être activés que depuis l'état Marche. Et une fois l'action terminée, la transition consistera à revenir à l'état Marche si besoin.

Ce choix peut paraître maladroit, car pour l'état Creuser par exemple, une fois terminé, le lemming devrait passer en état Chute fort probablement. Mais avec notre convention, il va d'abord passer en état Marche et juste après en état Chute. Cette petite erreur va durer 1/20eme de seconde et personne ne s'en apercevra et cette approximation nous fera économiser beaucoup de lignes de code.



L'ajout de nouveaux états se fait en « étoile » autours de l'état Marche. Cela évite d'arriver à un schéma trop complexe avec des dizaines d'états et de transitions à gérer, comme dans le diagramme ci-dessous :



Lilian BUZER

ESIEE PARIS

TODOLIST

Attention, vous devez respecter les trois étapes : gestion des états / gestions des actions et affichage. Il serait maladroit de coder l'affichage dans la gestion des changements d'état ou dans les actions.

- Faîtes en sorte qu'un total de 15 lemmings soit créés

- Testez la valeur du pixel sous le sprite du lemming, s'il n'est pas noir, vous devez changer l'état actuel pour Dead ou Marche

- Gérez l'affichage de l'état Marche

- Gérez l'action de l'état Marche

- L'ensemble des lemmings marchent au pas ! Normalement, les lemmings utilisent en même temps le même sprite de marche ce qui produit cet effet désagréable. Ajoutez un paramètre « Decal » à chaque lemming et initialisez-le par un nombre tiré au hasard. Servez-vous de ce nombre dans l'affichage pour que les lemmings aient un pas de marche désynchronisé !

Les lemmings devraient se déplacer jusqu'à la gauche, tomber de la falaise et s'écraser sur le sol.

- Gérez l'affichage de l'état Dead (dernière ligne en bas sur la planche de sprites). Cet état est terminal, il n'y a pas de transition pour en sortir !

- Gérer l'animation de l'état Dead. Attention, une fois l'animation jouée, elle ne recommence pas !

Nous allons programmer les premières modifications de comportement. Nous choisissons une tâche dans l'interface : creuser, stopper… en cliquant sur le bouton correspondant. Ensuite, lorsque nous cliquons sur un lemming, son état est modifié pour correspondre à la tâche demandée.

- Détectez le clic sur les boutons en bas de l'écran. La sélection d'une tâche entraîne l'allumage de la petite lampe présente en haut de chaque bouton, choisissez une couleur quelconque. Lorsque l'on clique sur un autre bouton, cela désactive le bouton précédent. Un seul bouton est actif.

- Nous gérons la tâche « STOP ». C'est un état terminal, impossible d'en sortir. Si l'outil « STOP » est actif et que le joueur clique sur un lemming, passez son action à STOP
    - ♣ Pour passer à STOP il faut que le lemming soit précédemment en mode MARCHE

- Maintenant, lorsqu'un lemming rencontre un lemming en mode STOP, il doit changer de direction. Programmez la détection de collision à gauche et à droite des lemmings. Maintenant, ils doivent changer de direction aussi lorsqu'ils rencontrent un mur

- ♣ Pour détecter la collision, il faut dessiner les lemmings en mode STOP avant l'étape 1, ils seront ainsi détectés comme un obstacle lors des tests
- ♣ Rien ne vous empêche de les réafficher dans l'étape 3 ☺
- ♣ Pour marcher vers la droite, utilisez la symétrie et un décalage pour caler le sprite

Il faut pouvoir gagner !!! Nous allons maintenant mettre en place l'action « creuser ». Elle nous permettra de creuser un puit jusqu'à la sortie.

- Si l'outil « Creuser » est actif et que le joueur clique sur un lemming, passez son état à Creuser
  - ♣ Pour passer à l'état Creuser, le lemming doit être dans l'état Marche
  - ♣Pour activer un état propre à un outil, il faut que le lemming soit en état marche auparavant

- Gérez l'animation de l'état Creuser

- Action de l'état « Creuser» : si le lemming creuse, toutes les 2 secondes, il enlève les 20 pixels sous son sprite. Il faut passer ses 20 pixels à la couleur noire. A ce moment-là, sa position descend d'un pixel

- Transition de l'état « Creuser » : si les 20 pixels sous le lemming sont noirs → mode Marche

- Normalement, vos lemmings devraient arriver entiers sur la ligne du bas. Positionnez le sprite de la sortie et dessinez-le juste avant l'affichage des lemmings pour qu'ils apparaissent par dessus.

- Détectez lorsqu'un lemming est proche de la porte et retirez-le de la liste courante

- Si vous avez sauvé au moins 10 lemmings, affichez « WIN » en gros au centre de l'écran

---

PROJET

Il est possible de continuer le jeu en tant que projet personnel guidé, voici les autres outils possibles : escalier, tunnel, bombe, pioche, parachute. Examinez quelques vidéos youtube pour bien comprendre la logique de chaque action.

Pour créer les décors, vous pouvez le faire en deux couleurs, pour vous faciliter la vie : noir pour le fond et marron les murs.