Donkey Kong Jr

Origine de l'idée du travail : jeu électronique Donkey Kong Jr (Game & Watch) de Nintendo.

1. Aperçu général du jeu

Donkey Kong Jr (DK Jr) doit libérer Donkey Kong (DK) qui a été capturé par Mario. Pour libérer DK, DK Jr doit atteindre, puis attraper la clé à 4 reprises pour ouvrir chaque partie de la cage. Sur son chemin, DK Jr doit éviter des ennemis (crocos et corbeaux). Chaque fois que DK Jr ouvre une partie de la cage, on gagne 10 poids. Quand les 4 parties de la cage sont ouvertes et que DK est libéré, on gagne 10 points additionnels. Ensuite, la cage se referme.

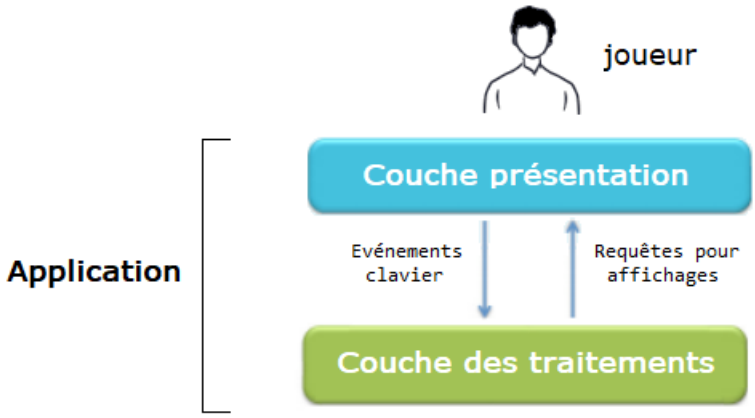
Quand DK Jr tombe dans le buisson après avoir tenté en vain d'attraper la clé, ou quand il entre en collision avec un ennemi (corbeau ou croco), il perd une vie. À ce moment, une tête de DK Jr apparaît à droite dans la zone Echecs, puis on peut jouer à nouveau. Lorsque 3 vies ont été perdues, la partie est terminée.

2. Architecture générale de l'application

Cette application repose sur une architecture en 2 couches :

- La **couche présentation**, pour le compte de la couche des traitements, crée la fenêtre du jeu, intercepte les événements produits par le joueur (frappe sur une touche du curseur, clic sur la croix de la fenêtre), et réalise les affichages des sprites.

- La **couche des traitements** gère tous les traitements relatifs au jeu : gestion de l'état de chaque sprite dans le jeu, prise en compte des collisions/interactions entre les sprites, et gestion du score.



Choisir une architecture en couches a un avantage important : quand on modifie l'implémentation interne d'une couche, mais que l'on conserve son interface, alors toute autre couche de l'application qui utilise les services de cette couche ne doit pas être modifiée.

La couche présentation est donnée aux étudiants. Le travail va consister à implémenter la couche des traitements.

3. La couche présentation

Les fichiers de code relatifs à la couche présentation se trouvent dans le dossier **/presentation**. Le fichier **presentation.c** contient les différentes fonctions, tandis que le fichier **presentation.h** donné ci-dessous contient le prototype de chacune de ces fonctions.

```c
#ifndef PRESENTATION_H
#define PRESENTATION_H

#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <SDL/SDL.h>

void ouvrirFenetreGraphique();

void afficherCle(int num);
void afficherCorbeau(int colonne, int num);
void afficherCroco(int colonne, int num);
void afficherDKJr(int ligne, int colonne, int num);
void afficherCage(int num);
void afficherRireDK();
void afficherEchec(int nbEchecs);
void afficherScore(int score);
void afficherChiffre(int ligne, int colonne, int chiffre);

void effacerCarres(int ligne, int colonne, int nbLignes = 1, int nbColonnes = 1);
void effacerPoints(int x, int y, int nbX = 1, int nbY = 1);

int lireEvenement();

#endif
```

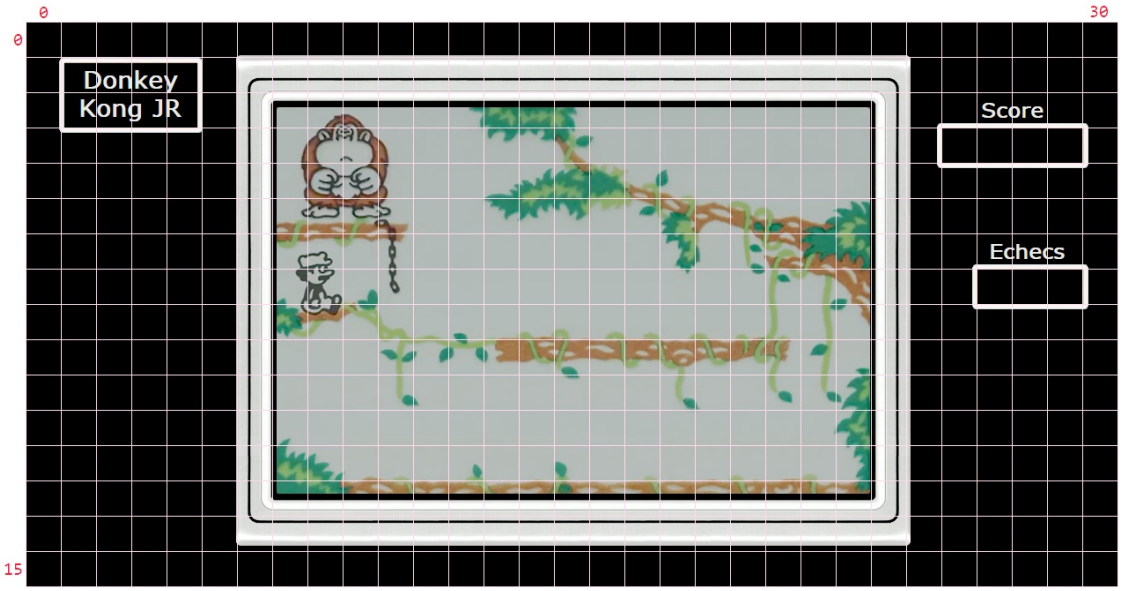
Description générale :

- La fonction **ouvrirFenetreGraphique**() charge toutes les images et crée la fenêtre graphique dans laquelle est affichée l'image du fond d'écran.

- La fonction **lireEvenement**() attend que le joueur appuie sur une touche du curseur ou clique sur la croix. En retour, elle fournit l'identifiant de la touche du curseur (SDLK_UP, SDLK_DOWN, SDLK_LEFT ou SDLK_RIGHT) sur laquelle a appuyé le joueur ou SDL_QUIT si le joueur a cliqué sur la croix.

- Les paramètres *ligne* et *colonne* pour les fonctions **afficherDKJr**(), **afficherCorbeau**(), **afficherCroco**(), **afficherChiffre**() correspondent aux coordonnées du carré à l'écran à partir d'où est réalisé l'affichage d'une image. Quand une fonction d'affichage, par exemple la fonction **afficherCle**(), ne demande pas de ligne et/ou de colonne en paramètre cela signifie que cette/ces information(s) est/sont déjà inscrite(s) dans la fonction.

  La fenêtre générale du jeu est découpée en carrés de 40x40 points. Le carré dans le coin supérieur gauche se trouve à la ligne 0 et à la colonne 0, tandis que le carré dans le coin inférieur droit est à la ligne 15 et à la colonne 30.



Les images des sprites sont elles-mêmes constituées d'un ou de plusieurs carrés de 40x40 points.

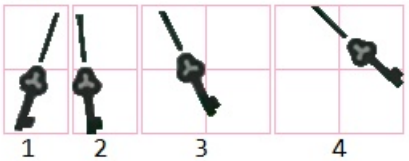
- La fonction **afficherChiffre**(), en plus des coordonnées d'affichage, attend un chiffre entre 0 et 9 en paramètre. Voici les images des chiffres :

- La fonction **afficherRireDK**() affiche l'image du rire aux coordonnées (3, 8) sur la tête de DK.

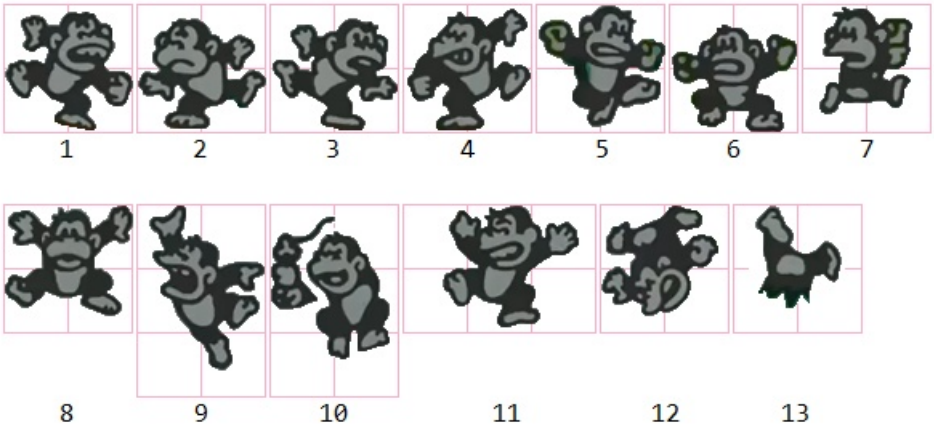
- La fonction **afficherEchec**() affiche la tête de DK Jr à droite. Cela permet de signaler la perte d'une vie. Le paramètre *num* correspond au nombre de vies déjà perdues.

- Le paramètre *num* pour les fonctions d'affichage **afficherCle**(), **afficherDKJr**(), **afficherCorbeau**(), **afficherCroco**() et **afficherCage**() correspond au numéro de l'image à afficher parmi plusieurs.

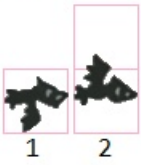
Clé:



DK Jr:

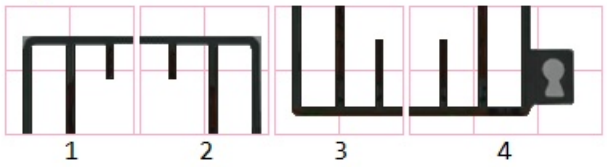


Corbeau:



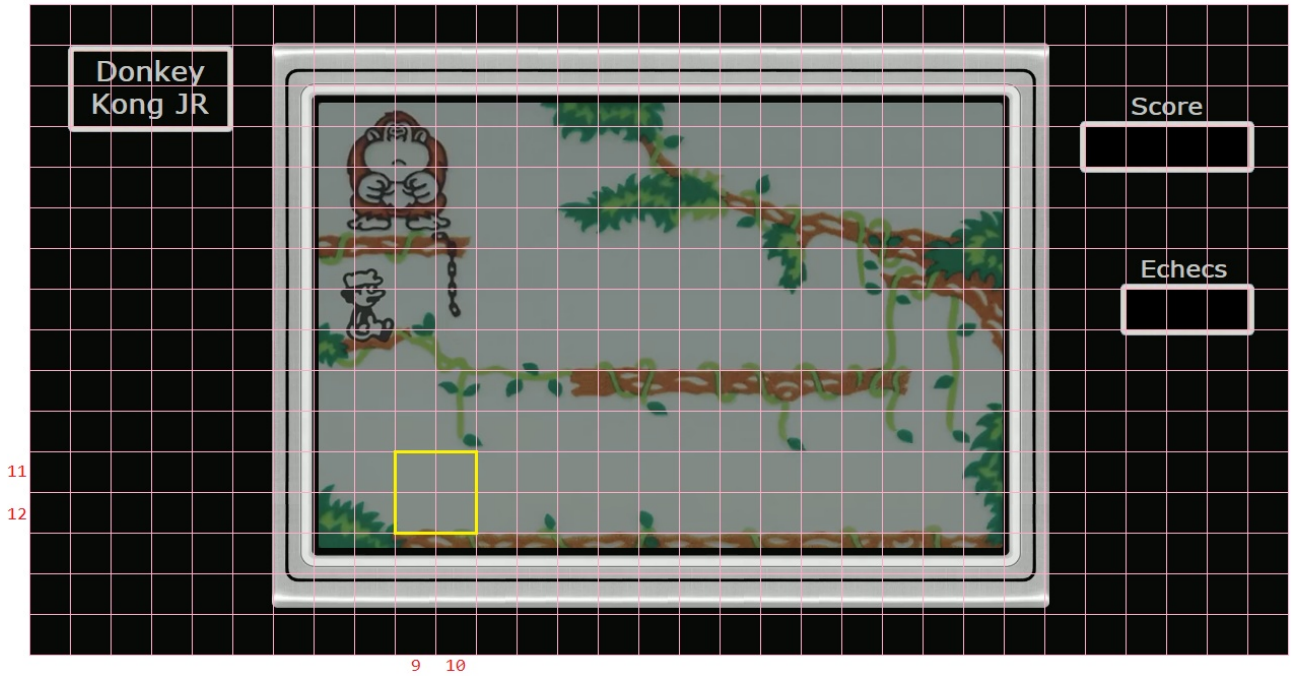
Croco:

Cage:



- La fonction **effacerCarres**() restaure l'image du fond d'écran sur une surface d'un ou de plusieurs carrés consécutifs de 40x40 points à partir de la ligne et de la colonne transmises en paramètre. Ceci a pour effet d'effacer l'image d'un sprite.

  Pour effacer, par exemple, DK Jr quand il se trouve en bas à gauche dans la fenêtre du jeu à partir de la ligne 11 et de la colonne 9, on utilisera effacerCarres() ainsi : `effacerCarres(11, 9, 2, 2)`. L'effacement portera sur la zone constituée des 4 carrés (11, 9), (11, 10), (12, 9), (12, 10).



- La fonction **effacerPoints**() restaure l'image du fond d'écran sur une surface d'un ou de plusieurs points à partir de coordonnées x et y en points fournies en paramètre.

Afin de comprendre comment utiliser les fonctions d'affichage de la couche présentation, supposons la fenêtre du jeu suivante :

Voici le code qui a produit cette fenêtre du jeu :

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
```

```c
#include <SDL/SDL.h>
#include "./presentation/presentation.h"

// ----------------------------------------------------------------------

int main(int argc, char* argv[])
{
	int evt;

	ouvrirFenetreGraphique();

	afficherCage(2);
	afficherCage(3);
	afficherCage(4);

	afficherRireDK();

	afficherCle(3);

	afficherCroco(11, 2);
	afficherCroco(17, 1);
	afficherCroco(0, 3);
	afficherCroco(12, 5);
	afficherCroco(18, 4);

	afficherDKJr(11, 9, 1);
	afficherDKJr(6, 19, 7);
	afficherDKJr(0, 0, 9);

	afficherCorbeau(10, 2);
	afficherCorbeau(16, 1);

	afficherEchec(1);

	afficherScore(1999);

	while(1)
	{
		evt = lireEvenement();

		switch(evt)
		{
		  case SDL_QUIT:
			exit(0);
		  case SDLK_UP:
			printf("KEY_UP\n");
			break;
		  case SDLK_DOWN:
			printf("KEY_DOWN\n");
			break;
		  case SDLK_LEFT:
			printf("KEY_LEFT\n");
			break;
		  case SDLK_RIGHT:
			printf("KEY_RIGHT\n");
		}
	}
}
```

Dans ce code, on voit comment est utilisée la fonction **lireEvenement**(). Cette fonction est bloquante. Elle retourne l'événement produit par le joueur qui peut être soit la frappe sur une touche spécifique du curseur (événement SDLK_UP, SDLK_DOWN, SDLK_LEFT ou SDLK_RIGHT), soit un clic sur la croix dans la fenêtre (événement SDL_QUIT).

4. La couche des traitements

4.1 Le tableau global grilleJeu

Le tableau global **grilleJeu** est un tableau à 2 dimensions constitué de 4 lignes et 8 colonnes qui va permettre de détecter les collisions pouvant survenir entre différents sprites dans le jeu : la clé, DK Jr, les crocos et les corbeaux.

```
S_CASE grilleJeu[4][8];

typedef struct
{
  int type;
  pthread_t tid;
} S_CASE;
```

Chaque cellule de grilleJeu est du type S_CASE :

- Le champ type contient le type du sprite. Il peut avoir comme valeur :

```
#define VIDE        0
#define DKJR        1
#define CROCO       2
#define CORBEAU     3
#define CLE         4
```

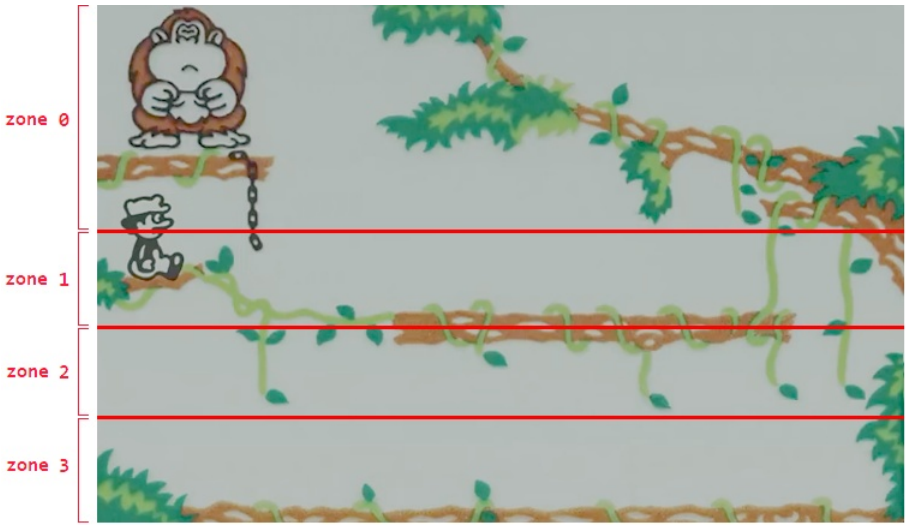
- Le champ tid contient l'identifiant du thread qui gère le sprite.

Pour manipuler grilleJeu, 3 fonctions sont données :

- `void initGrilleJeu()` : elle initialise le champ type à VIDE et le champ tid à 0 de toutes les cellules de grilleJeu.
- `void setGrilleJeu(int l, int c, int type = VIDE, pthread_t tid = 0)` : elle fixe le type d'un sprite dans le champ type et l'identifiant d'un thread dans le champ tid de la cellule située aux coordonnées (l, c) dans grilleJeu. Lors de l'appel à cette fonction, il est possible de ne passer aucune valeur en paramètre pour les champs type et tid. Les valeurs considérées sont alors, respectivement, VIDE et 0. Par exemple, setGrilleJeu(0, 1, CLE) fixe à CLE le champ type et à 0 le champ tid de la cellule grilleJeu[0][1], tandis que setGrilleJeu(0, 1) fixe à VIDE le champ type et à 0 le champ tid de cette même cellule.
- `void afficherGrilleJeu()` : elle affiche la valeur du champ type de toutes les cellules de grilleJeu. Cette fonction est pratique pour les débogages.

Tout accès à grilleJeu doit être protégé avec le mutex **mutexGrilleJeu**.

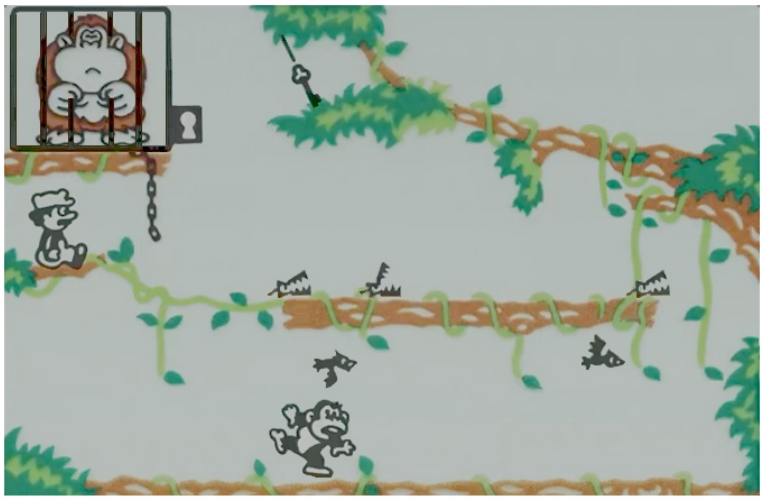
Pour faire le parallèle entre le tableau grilleJeu et l'écran du jeu, il faut savoir que celui-ci est découpé en 4 zones principales. Ces 4 zones sont celles dans lesquelles évoluent les sprites.



Dans la zone 0, on a la clé et DK Jr en haut quand il saute. Dans la zone 1, on a les crocos en haut et DK Jr quand il se déplace en haut horizontalement. Dans la zone 2, on a les corbeaux et DK Jr en bas quand il saute. Dans la zone 4, on a les crocos en bas et DK Jr quand il se déplace en bas horizontalement.

Les sprites présents dans la zone x à l'écran sont inscrits dans la ligne x de grilleJeu.

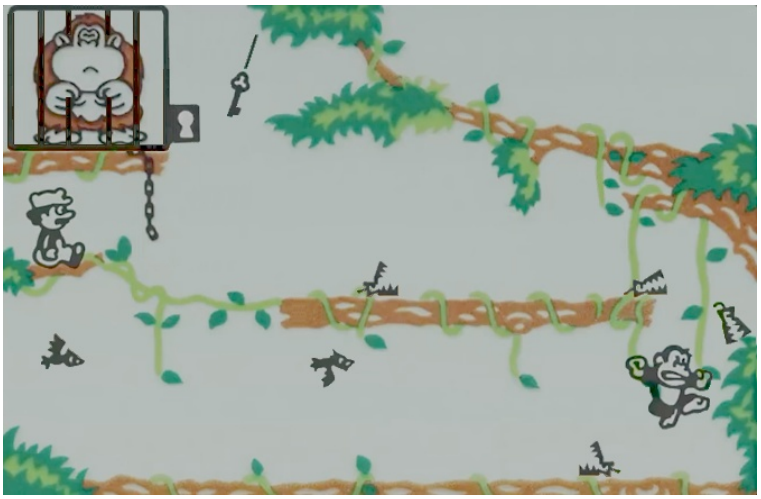
Exemple 1 :



DK Jr se trouve dans la zone 3, il y a 2 corbeaux dans la zone 2, il y a 3 crocos dans la zone 1, et la clé, dans la zone 0, est inaccessible pour DK Jr. Voici le champ type des cellules de grilleJeu :

```
0  0  0  0  0  0  0  0
0  0  0  2  2  0  0  2
0  0  0  3  0  0  3  0
0  0  0  1  0  0  0  0
```

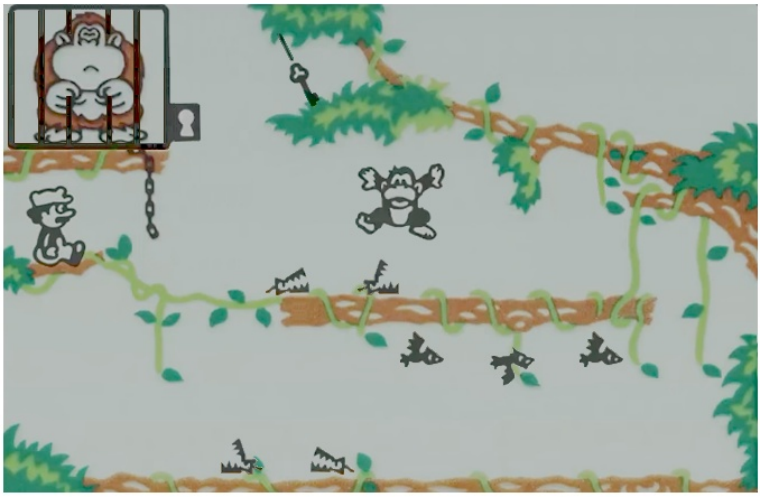
Exemple 2 :



DK Jr se trouve dans la zone 2, il y a 2 corbeaux dans la zone 2, il y a 2 crocos dans la zone 1 (le croco qui tombe ne compte pas) et 1 croco dans la zone 3, et la clé, dans la zone 0, est accessible pour DK Jr. Voici le champ type des cellules de grilleJeu :

```
0  4  0  0  0  0  0  0
0  0  0  0  2  0  0  2
3  0  0  3  0  0  0  1
0  0  0  0  0  0  2  0
```

Exemple 3 :



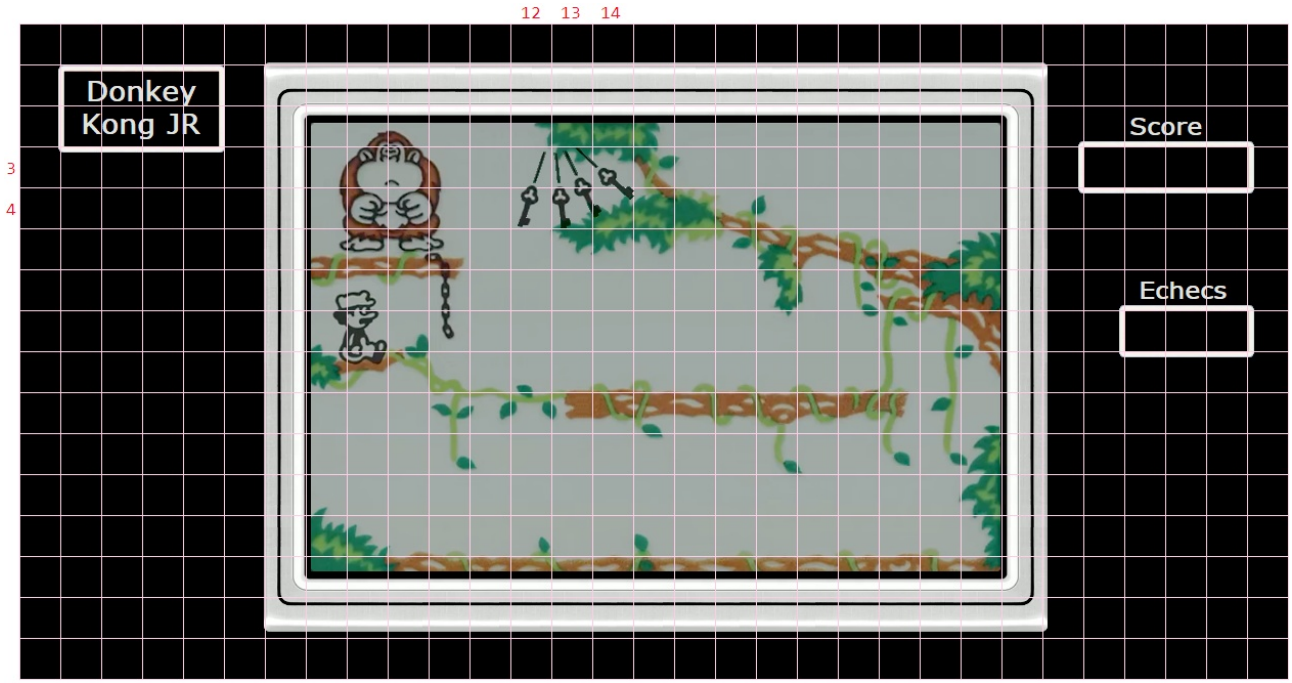
DK Jr se trouve dans la zone 0, il y a 3 corbeaux dans la zone 2, il y a 2 crocos dans la zone 1 et 2 crocos dans la zone 3, et la clé, dans la zone 0, est inaccessible pour DK Jr. Voici le champ type des cellules de grilleJeu :

```
0  0  0  0  1  0  0  0
0  0  0  2  2  0  0  0
0  0  0  0  3  3  3  0
0  0  2  2  0  0  0  0
```

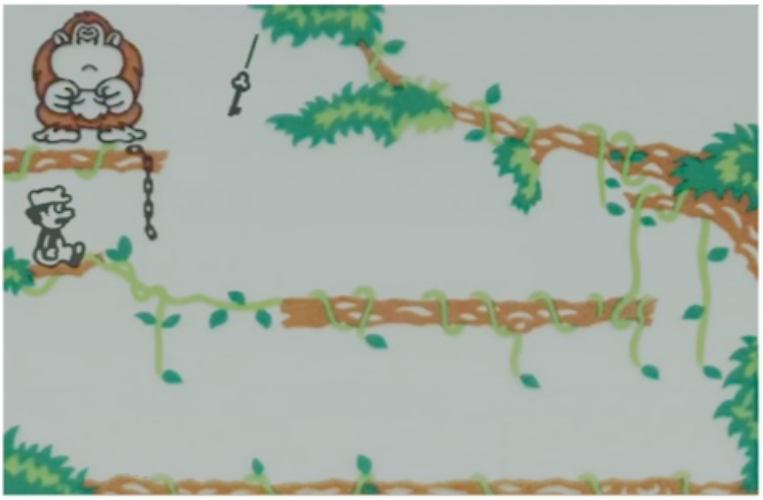
Voyons maintenant les étapes à suivre pour construire la couche des traitements du jeu. Afin de réaliser le jeu, il vous est demandé de suivre les étapes dans l'ordre et de respecter les contraintes d'implémentation citées, même si elles ne vous paraissent pas les plus appropriées (le but étant d'apprendre les techniques des threads et non d'apprendre à faire un jeu).

4.2 Étape 1 : ThreadCle

ThreadCle fait balancer la clé en haut à gauche près de DK. Il existe 4 positions pour la clé. Le passage d'une position à l'autre se fait toutes les 0,7 seconde.



Quand la clé est dans sa position la plus proche de DK, ThreadCle l'indique en plaçant la valeur CLE dans grilleJeu[0][1].type.



Dans les autres cas, la valeur VIDE se trouve dans grilleJeu[0][1].type.

La valeur de grilleJeu[0][1].type permettra à ThreadDKJr de savoir quand la clé sera accessible et quand elle ne le sera pas.

4.3 Étape 2 : ThreadEvenements

ThreadEvenements intercepte la frappe sur toute touche du curseur ainsi que le clic sur la croix dans la fenêtre pour terminer l'application. Il y a un exemple de code pour intercepter la frappe sur une touche du curseur ainsi que le clic sur la croix à la page 7 de ce document.

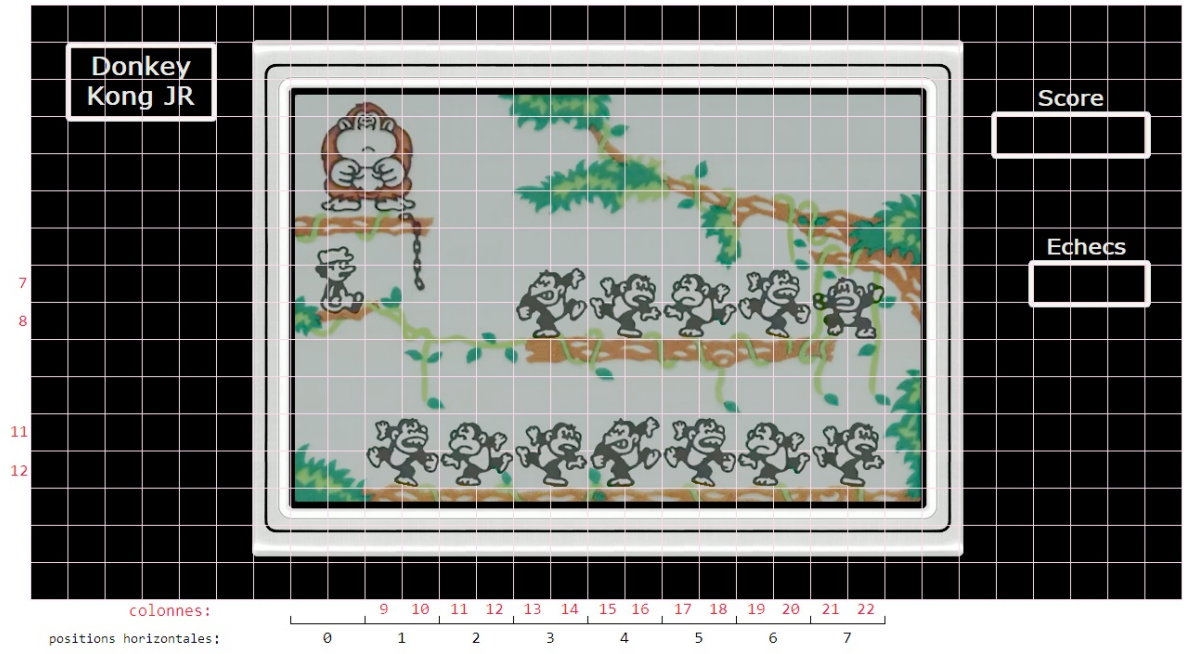
Quand aucun événément n'est produit par le joueur, la variable globale **evenement** contient la valeur AUCUN_EVENEMENT. Quand le joueur frappe sur une touche du curseur, ThreadEvenements stocke l'identifiant de cette touche (SDLK_UP, SDLK_DOWN, SDLK_LEFT ou SDLK_RIGHT) dans la variable evenement, puis il déclenche le signal **SIGQUIT** qui a pour but, comme on va le voir, de réveiller ThreadDKJr. Ensuite, ThreadEvenements attend pendant 100 millisecondes, il remet la valeur AUCUN_EVENEMENT dans la variable evenement, et il attend la survenue du prochain événement. Quand le joueur clique sur la croix, TheadEvenements récupère l'événement SDL_QUIT et termine l'application. La variable evenement est protégée avec le mutex **mutexEvenement**.

4.4 Étape 3 : ThreadDKJr

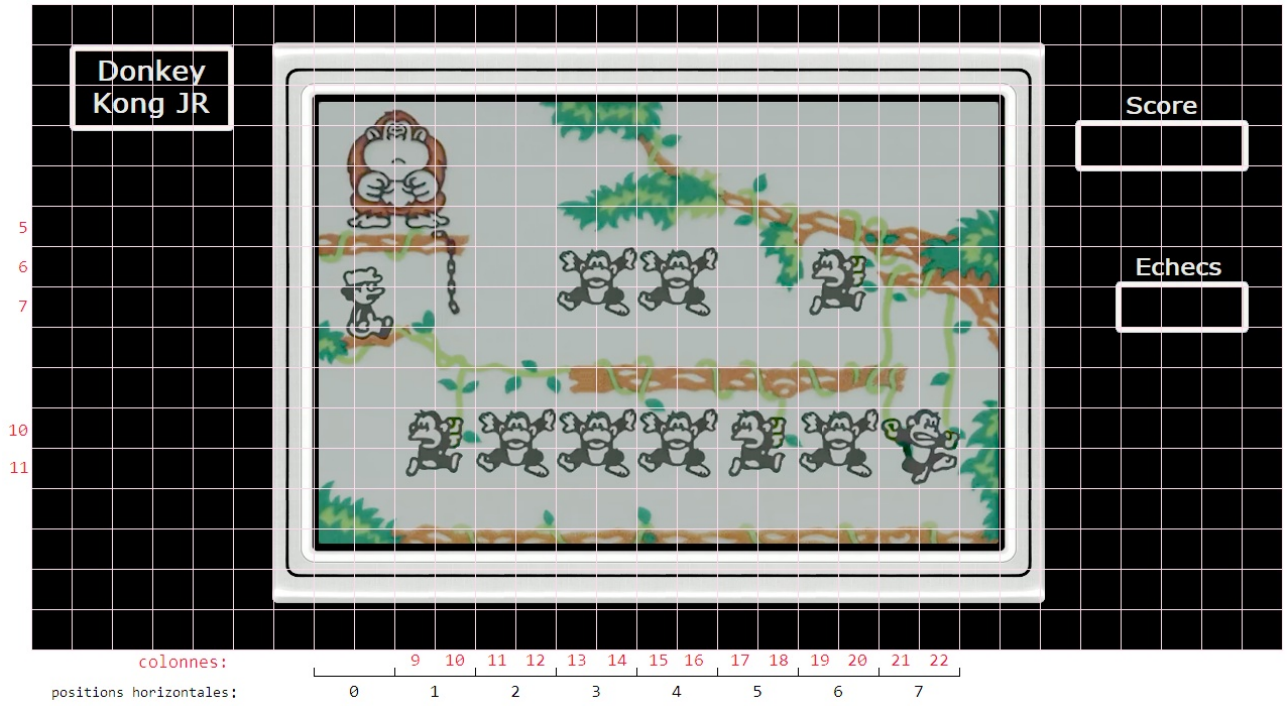
ThreadDKJr gère les déplacements de DK Jr qui est notre personnage dans le jeu.

Localisation de DK Jr à l'écran :

DK Jr se déplace horizontalement en haut dans les lignes 7 et 8 et les colonnes 13 à 22, et il se déplace horizontalement en bas dans les lignes 11 et 12 et les colonnes 9 à 22. DK Jr se déplace vers la gauche ou vers la droite de 2 colonnes à la fois.

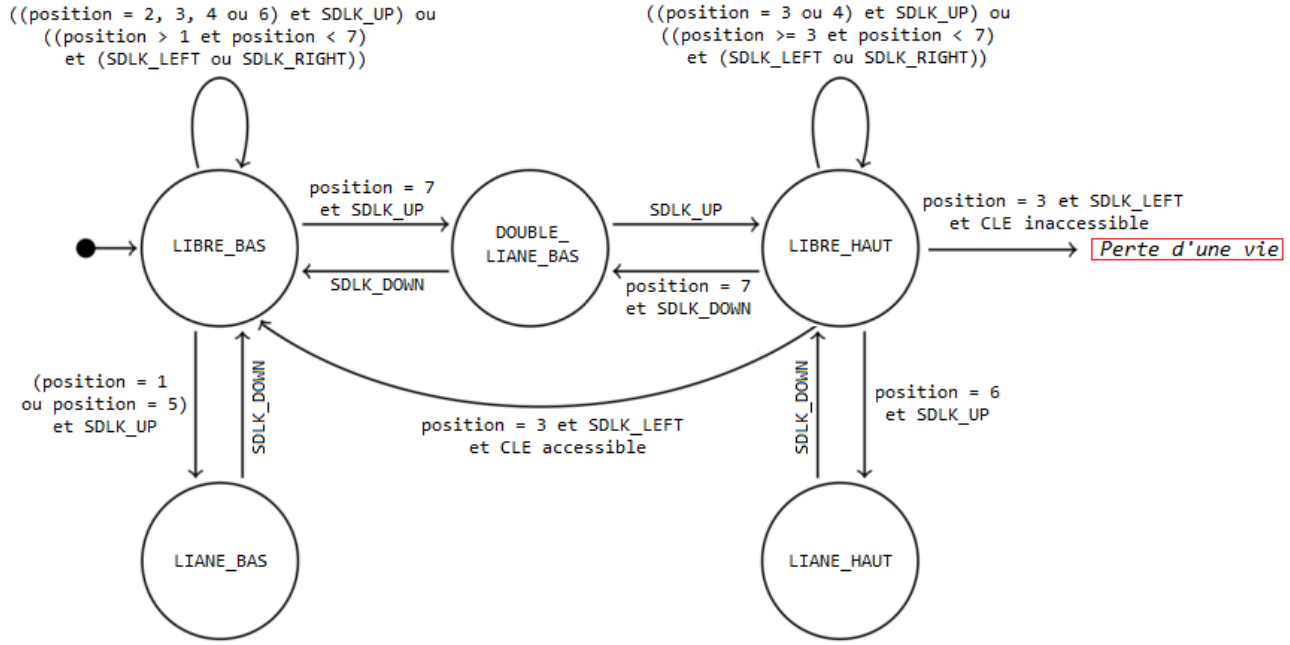


Quand il saute en haut, DK Jr se trouve dans les lignes 5, 6 et 7 et les colonnes 12 à 20, et, quand il saute en bas, DK Jr se trouve dans les lignes 10 et 11 et les colonnes 9 à 22.



États de DK Jr :

ThreadDKJr peut être vue comme implémentant une machine à états finis dans laquelle figure chaque état par lequel DK Jr passe à différentes reprises, durant une partie du jeu, et chaque transition permettant le passage d'un état à un autre. Voici les différents états par lesquels peut passer DK Jr :



position est la position horizontale de DK Jr. Les positions horizontales possibles sont reprises dans les 2 captures de la fenêtre du jeu données ci-avant.

À partir de ce diagramme, voici une idée de l'implémentation de ThreadDKJr :

```
void* FctThreadDKJr(void* p)
{
    bool on = true;

    pthread_mutex_lock(&mutexGrilleJeu);

    setGrilleJeu(3, 1, DKJR);
    afficherDKJr(11, 9, 1);
    etatDKJr = LIBRE_BAS;
    positionDKJr = 1;

    pthread_mutex_unlock(&mutexGrilleJeu);

    while (on)
    {
        pause();

        pthread_mutex_lock(&mutexEvenement);
        pthread_mutex_lock(&mutexGrilleJeu);

        switch (etatDKJr)
        {
            case LIBRE_BAS:
                switch (evenement)
                {
                     case SDLK_LEFT:
                         if (positionDKJr > 1)
                         {
                             setGrilleJeu(3, positionDKJr);
                             effacerCarres(11, (positionDKJr * 2) + 7, 2, 2);

                             positionDKJr--;

                             setGrilleJeu(3, positionDKJr, DKJR);
                             afficherDKJr(11, (positionDKJr * 2) + 7,
                                          ((positionDKJr - 1) % 4) + 1);
                         }
                         break;

                     case SDLK_RIGHT:
                         ...
                         break;

                     case SDLK_UP:
                         ...
                         break;
                }

            case LIANE_BAS:
                ...
                break;

            case DOUBLE_LIANE_BAS:
                ...
                break;

            case LIBRE_HAUT:
                ...
                break;

            case LIANE_HAUT:
                ...
```

```
        }

        pthread_mutex_unlock(&mutexGrilleJeu);
        pthread_mutex_unlock(&mutexEvenement);
    }

    pthread_exit(0);
}
```

ThreadDKJr attend l'arrivée du signal **SIGQUIT**. Ce signal est déclenché par ThreadEvenements quand le joueur appuie sur une touche du curseur. L'identifiant de la touche se trouve dans la variable globale **evenement** qui est protégée avec le mutex **mutexEvenement.**

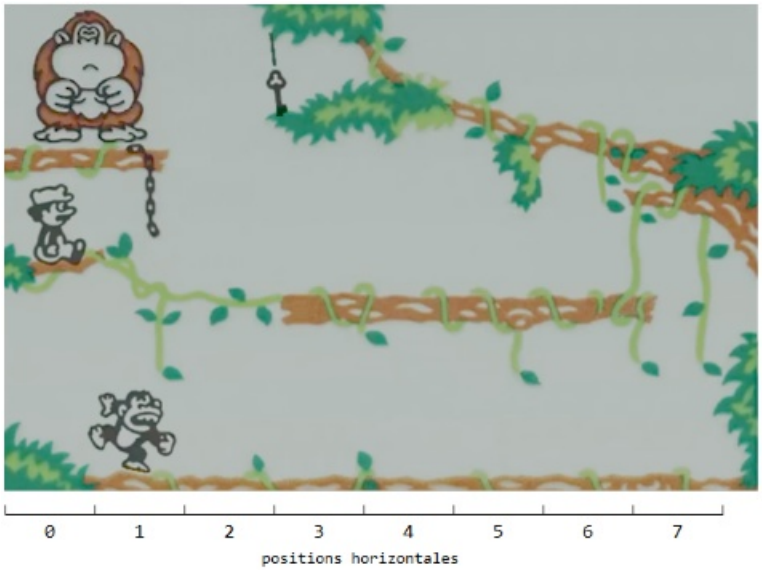
La variable globale **etatDKJr** stocke l'état courant de DK Jr. Cette variable est globale parce que, comme on le verra, elle sera aussi utilisée dans le handler du signal SIGINT. La position horizontale de DK Jr est stockée dans la variable globale **positionDKJr**. Cette variable est globale parce que, comme on le verra, elle sera aussi utilisée dans le handler des signaux SIGHUP, SIGCHLD, et SIGINT. En fonction de l'événement provenant de ThreadEvenements, ThreadDKJr détermine la position horizontale suivante ou l'état suivant de DK Jr.

Tant que l'événement produit par le joueur n'a pas été totalement traité dans ThreadDKJr, celui-ci conserve **mutexGrilleJeu** et mutexEvenement. Pour mutexGrilleJeu, il y a cependant une exception. Lorsque DK Jr saute sans s'accrocher à une liane ou à une double liane, son état ne change pas (selon qu'il se trouve en bas ou en haut, DK Jr demeure dans l'état LIBRE_BAS, ou dans l'état LIBRE_HAUT), mais DK Jr reste simplement en l'air pendant 1,4 seconde après quoi il revient sur le sol. Durant cette attente de 1,4 seconde, ThreadDKJr libère mutexGrilleJeu pour permettre aux autres threads de continuer à accéder au tableau grilleJeu, mais il conserve mutexEvenement.

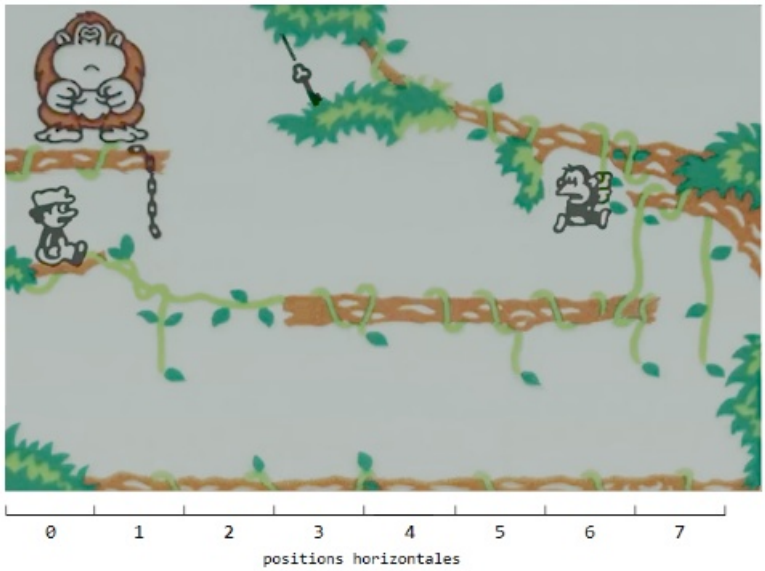
Localisation de DK Jr dans grilleJeu :

Pour rappel, l'écran du jeu est découpé en 4 zones (cf. page 9). L'état de DK Jr stocké dans la variable etatDKJr et la position horizontale de DK Jr stockée dans la variable positionDKJr permettent de savoir exactement où DK Jr se trouve dans grilleJeu.

Par exemple, dans l'écran suivant, DK Jr se trouve dans l'état LIBRE_BAS et à la position horizontale 1. Ces 2 informations permettent de savoir que la valeur DKJR se trouve dans grilleJeu[3][1].type.



Par exemple, dans l'écran suivant, DK Jr se trouve dans l'état LIANE_HAUT et à la position horizontale 6. Ces 2 informations permettent de savoir que la valeur DKJR se trouve dans grilleJeu[0][6].type.



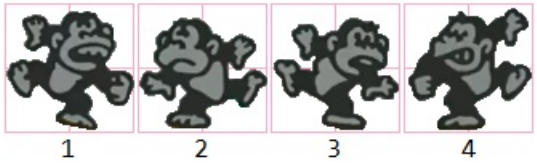
Pour revenir au code de ThreadDKJr partiellement donné ci-avant, on voit concrètement ce qui se passe quand DKJr est dans l'état LIBRE_BAS et que le joueur appuie sur la flèche gauche du curseur.

```
setGrilleJeu(3, positionDKJr);
effacerCarres(11, (positionDKJr * 2) + 7, 2, 2);

positionDKJr--;

setGrilleJeu(3, positionDKJr, DKJR);
afficherDKJr(11, (positionDKJr * 2) + 7, ((positionDKJr - 1) % 4) + 1);
```

On fixe à VIDE le champ type de la cellule grilleJeu[3][positionDKJr]. Ensuite, on efface l'image de DK Jr à l'écran avec effacerCarre(). Les paramètres passés à cette fonction sont : 11 est le numéro de la ligne dans la fenêtre, (positionDKJr * 2) + 7 transforme la position horizontale courante de DK Jr en un numéro de colonne à l'écran, et les valeurs 2 indiquent le nombre de carrés par colonne et par ligne à effacer. Après, on déplace DK Jr vers la gauche d'une position horizontale en décrémentant positionDKJr. Puis, on fixe à DKJR le champ type de la cellule grilleJeu[3][positionDKJr]. Enfin, on affiche DK Jr à sa nouvelle position. Au niveau des paramètres transmis à afficherDKJr(), on trouve ((positionDKJr - 1) % 4) + 1 qui sélectionne l'image de DK Jr à afficher parmi les 4 suivantes :



Gestion des 3 vies :

Le thread principal crée ThreadDKJr et attend la fin de son exécution avec **pthread_join**(). Quand DK Jr est en haut en position horizontale 3 et qu'on appuie sur la flèche gauche, il saute pendant 0,5 seconde, puis il tombe dans le buisson et ThreadDKJr se termine. Le thread principal est alors réveillé. Pour signaler au joueur la perte de la vie, le thread principal affiche une tête de DK Jr à droite de l'écran. Si le nombre de vies perdues est inférieur à 3, le thread principal recrée ThreadDKJr, sinon la partie est terminée, le thread principal se met en pause et la seule opération que peut encore faire le joueur est de cliquer sur la croix dans le coin supérieur droit de la fenêtre. Quand cet événement survient, ThreadEvenements termine l'application.

À présent, on peut voir la clé balancer et, en même temps, on peut déplacer DK Jr en haut et en bas. Voyons maintenant comment faire pour permettre à DK Jr d'attraper la clé, d'ouvrir la cage de DK et d'obtenir des points ...

4.5 Étape 4 : ThreadDK

ThreadDK gère l'ouverture des 4 parties de la cage dans laquelle est enfermé DK. Au départ, les 4 parties de la cage sont fermées. ThreadDK attend sur la variable de condition **condDK**. Lorsque DK Jr est en haut (état LIBRE_HAUT) et en position horizontale 3 et que le joueur frappe sur la touche gauche du curseur, DK Jr saute vers la gauche pour attraper la clé. À ce moment, 2 cas sont possibles :

- La valeur CLE se trouve dans grilleJeu[0][1].type indiquant que la clé est accessible : DK Jr attrape alors la clé et ouvre une partie de la cage. ThreadDKJr place la valeur true dans la variable **MAJDK** et réveille ThreadDK avec pthread_cond_signal() sur la variable de condition condDK. ThreadDK est réveillé et, comme la valeur true se trouve dans MAJDK, il efface la partie de la cage qui est en cours d'ouverture. Si la cage est complètement ouverte, alors DK rigole pendant 0,7 seconde, puis la cage se referme complètement. Ensuite, DK Jr retrouve sa place initiale en bas à gauche.

  La variable MAJDK est protégée avec le mutex **mutexDK**.

- La valeur VIDE se trouve dans grilleJeu[0][1].type indiquant que la clé est inaccessible : DK Jr ne peut alors attraper la clé et, comme il tombe dans le buisson, il perd une vie. Ensuite, si la partie n'est pas finie, DK Jr retrouve sa place initiale en bas à gauche.

Durant l'ouverture d'une partie de la cage ou la chute de DK Jr dans le buisson, ThreadDKJr conserve **mutexGrille**. Ceci a pour effet de faire attendre les autres sprites du jeu.

4.6 Étape 5 : ThreadScore

ThreadScore gère l'affichage du score. Les variables **score** et **MAJScore** sont globales. Elles sont protégées avec le mutex **mutexScore**. Le score est modifié à 2 endroits différents :

- Lorsque DK Jr ouvre une partie de la grille de la cage, ThreadDKJr augmente le contenu de la variable score de 10 et assigne à la variable MAJScore la valeur true, puis il réveille ThreadScore qui était en attente sur la variable de condition **condScore**.

- Lorsque la grille est totalement ouverte et que DK rigole, ThreadDK augmente le contenu de la variable score de 10 et assigne à la variable MAJScore la valeur true, puis il réveille ThreadScore qui était en attente sur la variable de condition condScore.

Au moment de son réveil, ThreadScore teste la valeur de MAJScore pour vérifier que le score a bien changé, et, si c'est le cas, il affiche à droite le nouveau score figurant dans la variable score.

À présent, on peut complètement jouer une partie de Donkey Kong Jr. On peut déplacer DK Jr en bas et en haut, on peut lui faire attraper la clé et libérer DK, le score est mis à jour, et, lorsque DK Jr échoue à attraper la clé, il tombe dans le buisson et perd une vie. Il nous reste à ajouter les ennemis : corbeaux et crocos, dans le jeu...

4.7 Étape 6 : ThreadEnnemis

ThreadEnnemis génère un nouvel ennemi qui peut être un corbeau ou un croco à intervalles réguliers. Le délai d'attente avant la génération d'un nouvel ennemi est fixé au départ à 4 secondes dans la variable globale **delaiEnnemis**. Le choix de l'ennemi est aléatoire, cela peut être un corbeau ou un croco. Si l'ennemi choisi est un corbeau, alors ThreadEnnemis crée ThreadCorbeau pour gérer ce corbeau. Si l'ennemi choisi est un croco, alors ThreadEnnemis crée ThreadCroco pour gérer ce croco.

La variable delaiEnnemis est globale parce qu'elle est utilisée aussi dans le handler du signal SIGALRM. La difficulté du jeu augmente avec le temps. Toutes les 15 secondes, le signal SIGALRM est envoyé à ThreadEnnemis. Cette alarme est initialisée au départ dans ThreadEnnemis. Dans le handler de SIGALRM, on diminue de 0,25 seconde le délai stocké dans la variable delaiEnnemis, puis on réinitialise l'alarme à 15 secondes. On cesse de réinitialiser l'alarme dans le handler de SIGALRM quand la variable delaiEnnemis stocke un délai de 2,5 secondes qui est le délai minimum.

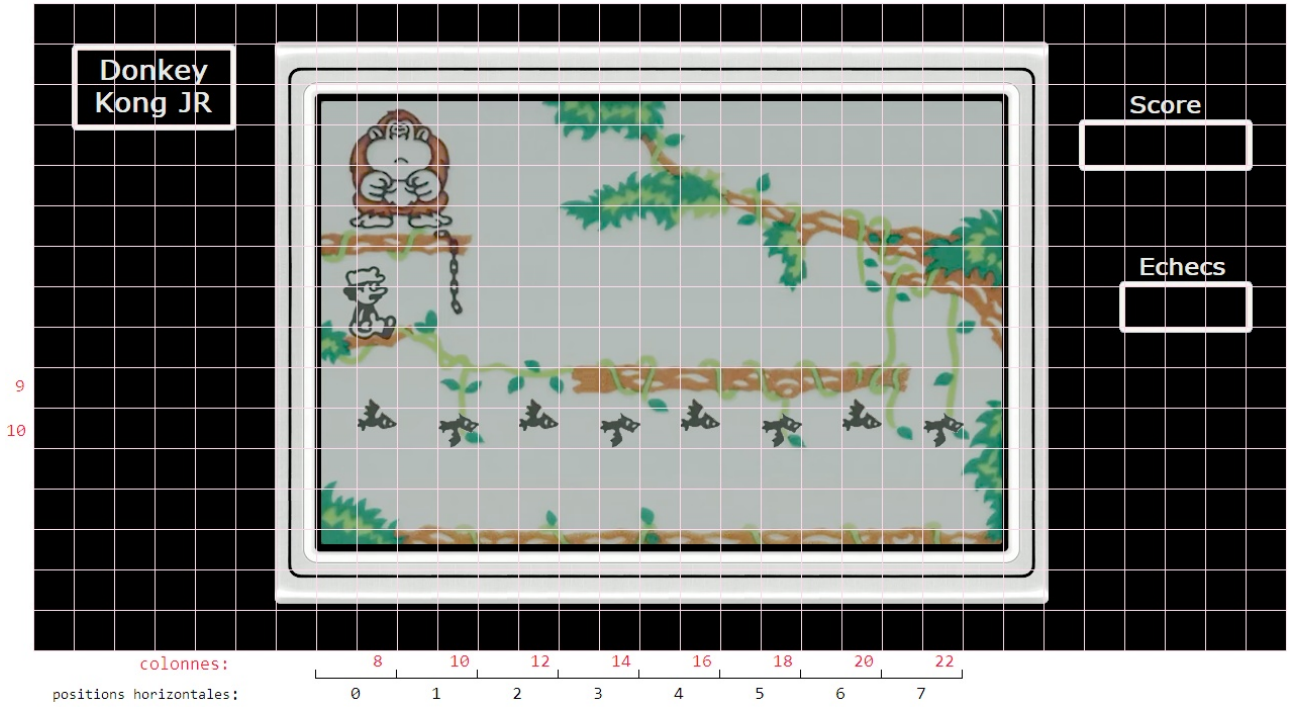
À ce stade, pour tester si ThreadEnnemis fonctionne correctement, vous pouvez afficher un message dans la console lorsque doit être créé un croco et un autre message lorsque doit être créé un corbeau.

4.8 Étape 7 : ThreadCorbeau

ThreadCorbeau est créé par ThreadEnnemis et gère le déplacement d'un corbeau spécifique. Le corbeau change de position horizontale toutes les 0,7 seconde. Il y a autant de TheadCorbeau créés par ThreadEnnemis qu'il y a de corbeaux visibles à l'écran. Dès qu'un corbeau a terminé son parcours, ThreadCorbeau se termine.

Localisation d'un corbeau à l'écran :

Un corbeau se déplace horizontalement dans les lignes 9 et 10 et les colonnes 8 à 22. Il se déplace de 2 colonnes à la fois. On stocke la position horizontale courante du corbeau dans une variable allouée dynamiquement et son adresse doit être rangée dans une **variable spécifique**. C'est à partir de cette position horizontale courante du corbeau qu'est déterminée la colonne à l'écran dans laquelle est fait l'affichage du corbeau (colonne à l'écran = position horizontale * 2 + 8).

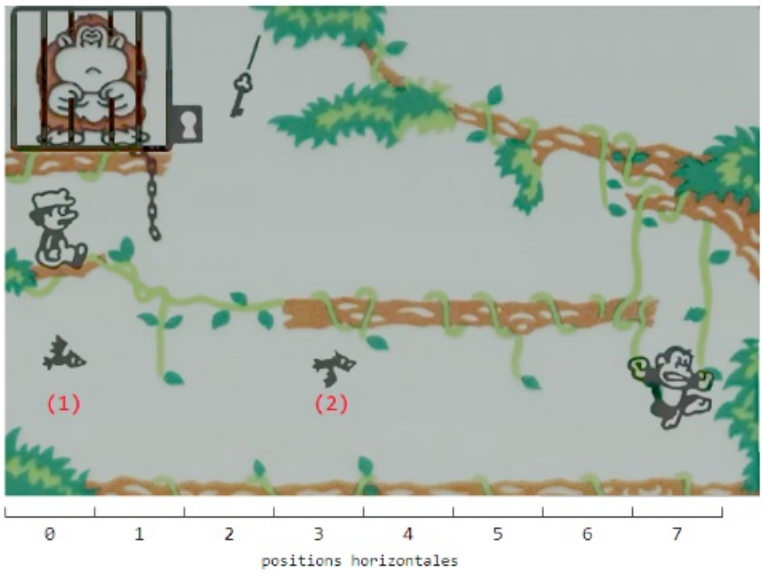


La position horizontale d'un corbeau est stockée dans une variable spécifique parce qu'elle est utilisée le handler du signal SIGUSR1.

Localisation d'un corbeau dans grilleJeu :

Stocker la position horizontale du corbeau permet de conserver le numéro de la cellule dans la ligne 2 de grilleJeu dans laquelle se trouve le corbeau.

Par exemple, dans l'écran du jeu suivant, on a 2 corbeaux :



Voici ce qu'on trouve au sujet de ces corbeaux dans grilleJeu :

- grilleJeu[2][0].type = CORBEAU et grilleJeu[2][0].tid = id du ThreadCorbeau qui gère le corbeau (1).

- grilleJeu[2][3].type = CORBEAU et grilleJeu[2][3].tid = id du ThreadCorbeau qui gère le corbeau (2).

Gestion des collisions :

La gestion des collisions est possible grâce aux informations stockées dans grilleJeu. Deux cas sont possibles et ils nécessitent la mise à jour de ThreadDKJr et de ThreadCorbeau :

- Collision entre DK Jr et un corbeau : quand DK Jr est dans l'état LIBRE_BAS et que le joueur appuie sur la flèche vers le haut du curseur, ThreadDKJr vérifie dans grilleJeu si aucun corbeau n'est présent dans la ligne 2 à cette position horizontale. Si c'est le cas, ThreadDKJr envoie le signal **SIGUSR1** au ThreadCorbeau qui gère ce corbeau, puis il se termine et on perd une vie. Le handler de SIGUSR1 utilise l'adresse de la zone de mémoire stockée dans la **variable spécifique** pour obtenir la position horizontale du corbeau. Ce handler efface le corbeau dans grilleJeu et à l'écran, puis il termine ThreadCorbeau.

- Collision entre un corbeau et DK Jr : quand un corbeau est sur le point de se déplacer d'une position horizontale vers la droite, ThreadCorbeau vérifie dans grilleJeu si DK Jr se trouve à la position horizontale dans laquelle doit être placé ce corbeau. Si c'est le cas, ThreadCorbeau envoie le signal **SIGINT** à ThreadDKJr, puis il se termine. Le handler de SIGINT efface DK Jr de l'écran (sa position horizontale est stockée dans **positionDKJr**), puis il termine ThreadDKJr et on perd une vie. Notons que si, au moment où le corbeau percute DK Jr, celui-ci est dans l'état LIBRE_BAS (état stocké dans **etatDKJr**), c'est-à-

dire que DKJr se trouve en l'air, mais sans s'être accroché à une liane ou à une double liane, il faut libérer **mutexEvenement** dans le handler de SIGINT pour ne pas bloquer à l'infini ThreadEvenements.

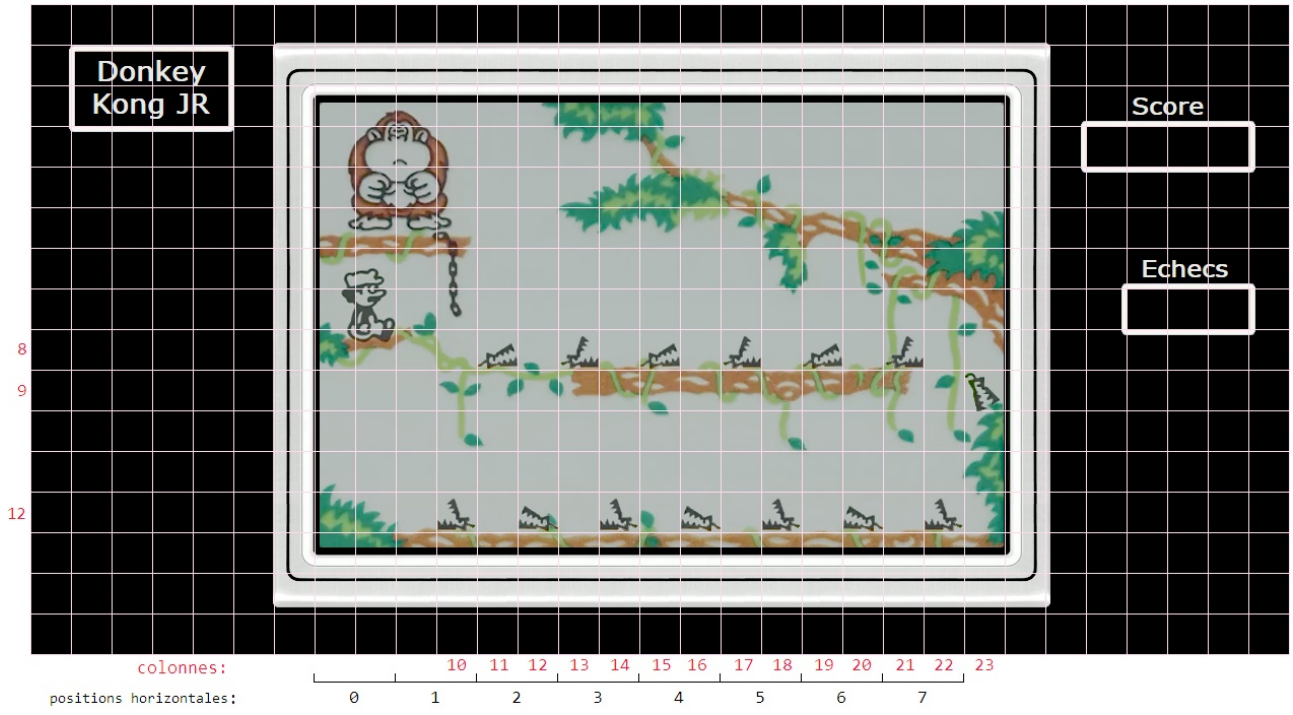
Le "destructeur" de la variable spécifique libère la mémoire allouée dynamiquement dans ThreadCorbeau.

4.9 Étape 8 : ThreadCroco

ThreadCroco est créé par ThreadEnnemis et gère le déplacement d'un croco spécifique. Le croco change de position horizontale toutes les 0,7 seconde. Il y a autant de ThreadCroco créés par ThreadEnnemis qu'il y a de crocos visibles à l'écran. Dès qu'un croco a terminé son parcours, ThreadCroco se termine.

Localisation d'un croco à l'écran :

Un croco se déplace horizontalement en haut dans les lignes 8 et 9 et les colonnes 11 à 23, et il se déplace horizontalement en bas dans la ligne 12 et les colonnes 22 à 10. Il se déplace de 2 colonnes à la fois.



Voici un nouveau type de données qui va permettre de sauver la position horizontale d'un croco ainsi que s'il se trouve en haut ou en bas :

```
typedef struct
{
	bool haut;
	int position;
} S_CROCO;
```

Le champ haut contient true si le croco est en haut (ligne 1 de grilleJeu), sinon ce champ contient false indiquant que le croco est en bas (ligne 3 de grilleJeu). Le champ position stocke la position

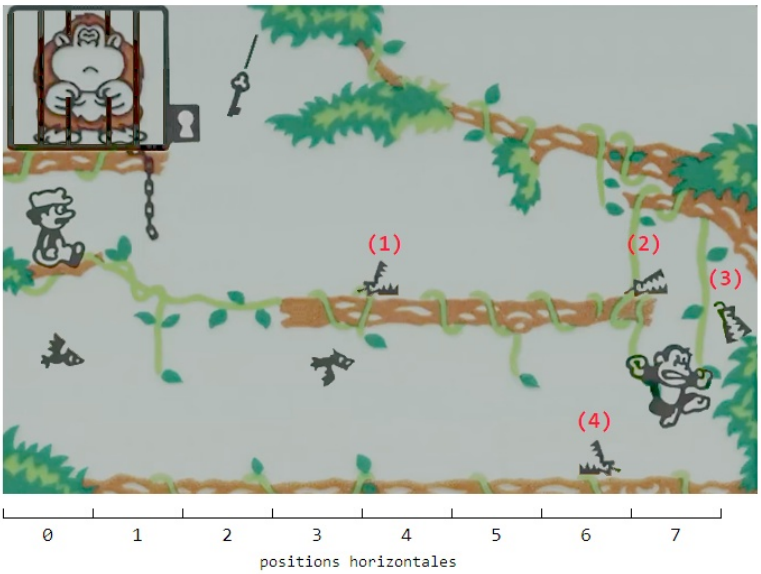
horizontale courante du croco. C'est à partir de cette position horizontale du croco qu'est déterminée la colonne à l'écran dans laquelle se fait l'affichage du croco en haut (colonne à l'écran = position horizontale * 2 + 7), ou en bas (colonne à l'écran = position horizontale * 2 + 8).

Une variable du type S_CROCO doit être allouée dynamiquement en mémoire et son adresse doit être rangée dans une **variable spécifique**, car elle est utilisée dans le handler du signal SIGUSR2.

Localisation d'un croco dans grilleJeu :

Les champs haut et position dans la structure S_CROCO permettent de connaître exactement où se trouve le croco dans grilleJeu.

Par exemple, dans l'écran du jeu suivant, on a 4 crocos :



Voici ce qu'on trouve au sujet de ces crocos dans grilleJeu :

- grilleJeu[1][4].type = CROCO et grilleJeu[1][4].tid = id du ThreadCroco qui gère le croco (1).
- grilleJeu[1][7].type = CROCO et grilleJeu[1][7].tid = id du ThreadCroco qui gère le croco (2).
- Le croco (3) ne peut pas entrer en collision avec DK Jr à l'endroit où il se trouve, car il est en train de passer du haut vers le bas. À cet instant, il n'y a aucune information à son sujet de stockée dans grilleJeu.
- grilleJeu[3][6].type = CROCO et grilleJeu[3][6].tid = id du ThreadCroco qui gère le croco (4).

Gestion des collisions :

La gestion des collisions est possible grâce aux informations stockées dans grilleJeu. Trois cas sont possibles et ils nécessitent la mise à jour de ThreadDKJr et de ThreadCroco :

- Collision entre DK Jr et un croco : quand DK Jr revient au sol, ThreadDKJr vérifie dans grilleJeu si aucun croco n'est présent à sa position horizontale. Si c'est le cas, ThreadDKJr envoie le signal **SIGUSR2** au ThreadCroco qui gère ce croco, puis il se termine et on perd une vie. Le handler de SIGUSR2 utilise l'adresse de la zone de mémoire stockée dans la **variable spécifique** pour obtenir la position complète du croco. Ce handler efface le croco dans grilleJeu et à l'écran, puis il termine ThreadCroco.

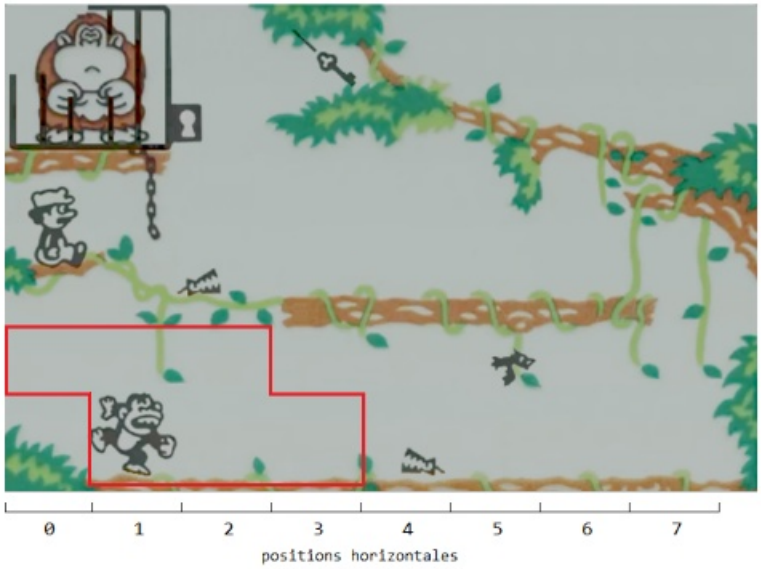
- Collision entre un croco et DK Jr en haut : quand un croco se déplace en haut d'une position horizontale vers la droite, ThreadCroco vérifie dans grilleJeu si DK Jr se trouve à la position horizontale dans laquelle doit être placé ce croco. Si c'est le cas, ThreadCroco envoie le signal **SIGHUP** à ThreadDKJr, puis il se termine. Le handler de SIGHUP efface DK Jr (il sait la position horizontale de DK Jr grâce à la variable globale **positionDKJr**), puis il termine ThreadDKJr et on perd une vie.

- Collision entre un croco et DK Jr en bas : quand un croco se déplace en bas d'une position horizontale vers la gauche, ThreadCroco vérifie dans grilleJeu si DK Jr se trouve à la position horizontale dans laquelle doit être placé ce croco. Si c'est le cas, ThreadCroco envoie le signal **SIGCHLD** à ThreadDKJr, puis il se termine. Le handler de SIGCHLD efface DK Jr (il sait la position horizontale de DK Jr grâce à la variable globale **positionDKJr**), puis il termine ThreadDKJr et on perd une vie.

Le "destructeur" de la variable spécifique libère la mémoire allouée dynamiquement dans ThreadCroco.

Voici un dernier point à implémenter :

Au moment où DK Jr retrouve sa place de départ en bas à gauche après avoir ouvert une des serrures de la cage ou après avoir perdu une vie, il ne doit pas entrer en collision avec un ennemi qui serait déjà présent dans les environs. C'est pourquoi tous les ennemis à proximité de la place de départ de DK Jr doivent être éliminés (ThreadDKJr envoie SIGUSR1 aux ThreadCorbeau concernés et SIGUSR2 aux ThreadCroco concernés) avant que n'apparaisse DK Jr. Cette élimination concerne les éventuels crocos en bas aux positions horizontales 1, 2, 3, ainsi que les éventuels corbeaux aux positions horizontales 0, 1, 2.



À présent, l'implémentation du jeu est terminée. Ceux qui veulent aller plus loin peuvent implémenter le bonus ...

4.10 Étape 9 : Bonus

Augmenter de 1 le score lorsque DK Jr saute avec succès au-dessus d'un croco.

Modifier ThreadScore afin de, tous les 300 points, récupérer une vie et remettre le délai d'attente utilisé par ThreadEnnemis avant la génération d'un nouvel ennemi à 3 secondes. Comme la variable globale **delaiEnnemis** est, à présent, utilisée par 2 threads, il faut veiller à la protéger avec un nouveau mutex.

4.11 Aperçu global du thread principal

Voici ce que fait le thread principal (la fonction main() dans le programme) :

1. Il commence par ouvrir la fenêtre graphique.

2. Il fixe à VIDE le champ type et à 0 le champ tid de toutes les cellules du tableau global grilleJeu.

3. Il associe un handler aux signaux :

   - SIGQUIT (déclenché par ThreadEvenements et réceptionné par ThreadDKJr)
   - SIGALRM (réceptionné par ThreadEnnemis)
   - SIGUSR1 (déclenché par ThreadDKJr et réceptionné par ThreadCorbeau)
   - SIGUSR2 (déclenché par ThreadDKJr et réceptionné par ThreadCroco)
   - SIGINT (déclenché par ThreadCorbeau et réceptionné par ThreadDKJr)
   - SIGHUP (déclenché par ThreadCroco et réceptionné par ThreadDKJr)
   - SIGCHLD (déclenché par ThreadCroco et réceptionné par ThreadDKJr)

4. Il initialise les variables de condition :

   - condDK (utilisée par ThreadDKJr pour réveiller ThreadDK)
   - condScore (utilisée par ThreadDKJr et par ThreadDK pour réveiller ThreadScore)

5. Il initialise les mutexes :

   - mutexGrilleJeu (utilisé pour protéger le tableau grilleJeu)
   - mutexDK (utilisé pour protéger la variable MAJDK)
   - mutexEvenement (utilisé pour protéger la variable evenement)
   - mutexScore (utilisé pour protéger les variables MAJScore et score)

6. Il crée la variable spécifique keySpec (utilisée par ThreadCorbeau pour fournir la position du corbeau au handler du signal SIGUSR1 et par ThreadCroco pour fournir la position du croco au handler du signal SIGUSR2). Dans les différents ThreadCroco et ThreadCorbeau, la variable spécifique stockera l'adresse d'une zone de mémoire allouée dynamiquement. Le destructeur de la variable spécifique aura pour fonction de libérer cette zone de mémoire allouée dynamiquement.

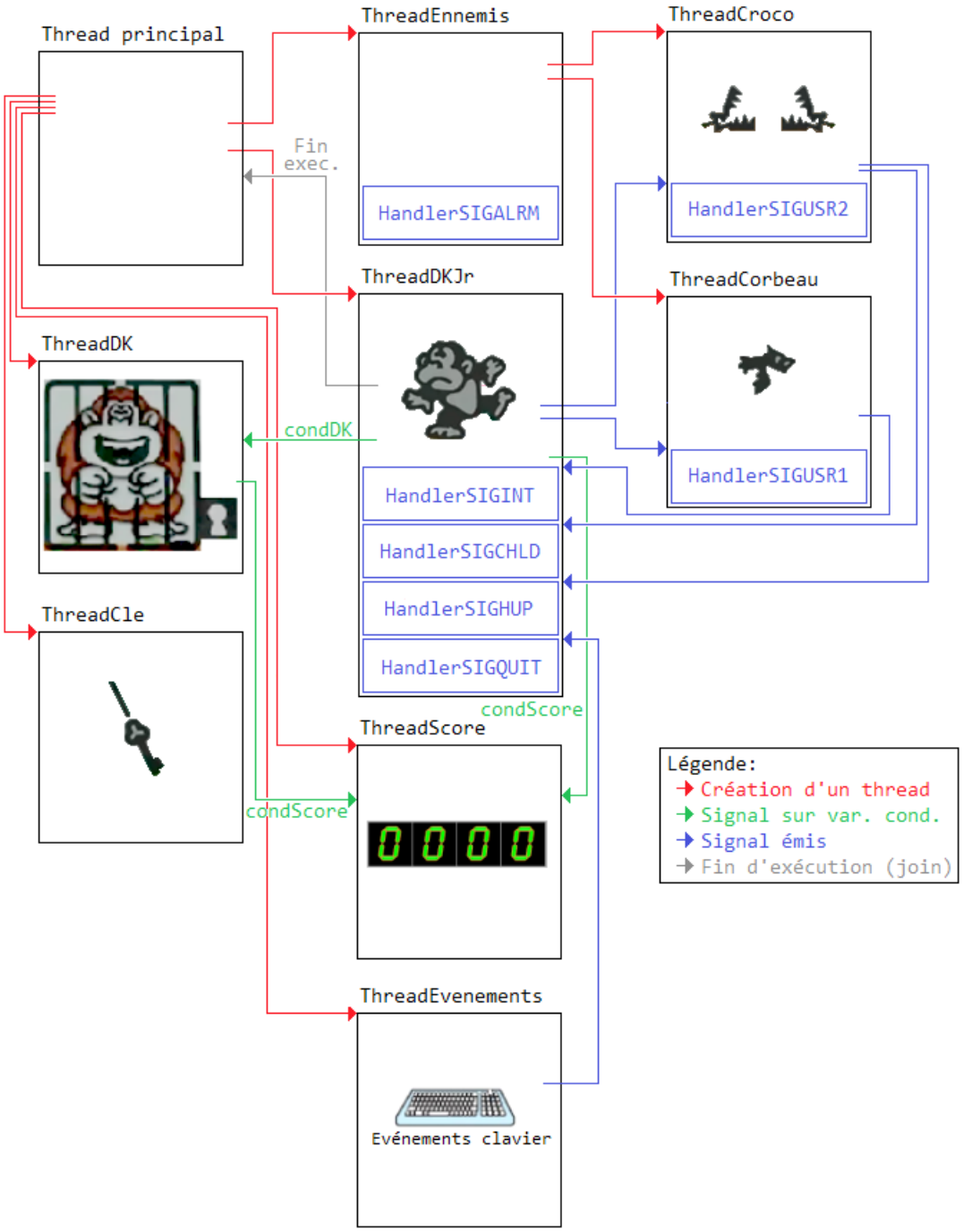
7. Il démarre les threads suivants :

   - threadCle
   - threadDK
   - ThreadEvenements
   - threadScore
   - threadEnnemis (il crée les threads ThreadCroco et ThreadCorbeau)
   - threadDKJr

8. Il attend avec pthread_join() la terminaison de ThreadDKJr et recrée ce thread si le nombre de vies perdues est inférieur à 3.

Concernant les signaux, il faut bien veiller à bloquer et débloquer les signaux correctement dans les différents threads.

4.12 Architecture de la couche des traitements



5. Consignes générales

Ce dossier doit être réalisé par groupe de 2 étudiants (ou 1 étudiant). Il devra être terminé pour une date donnée et l'archive contenant la solution devra être déposée à un endroit spécifique. Ces informations vous seront données par votre professeur. Le jour de l'évaluation, vous présenterez votre dossier oralement.