Rendu Snake MIPS

Le jeu comporte toutes les fonctionnalités requises au bon fonctionnement telles que la mise à jour de la direction (majDirecton), la mise à jour des structures de données (updateGameStatus), détection des conditions de fin de jeu (conditionFinJeu) et l'affichage de fin de partie dans le terminal(affichageFinJeu). On n'a pas modifié le déroulement du jeu. À cela nous avons ajouté les fonctionnalités supplémentaire bonus, donc l'affichage du score est graphique en plus d'être dans le terminal, le Rainbow Snake avec un tableau de couleurs qui boucle et un système de niveau prédéfinies par le niveau de sommeil du jeu qui diminue pour chaque bonbon mangé (limite basse définies).

En ce qui concerne les difficultés rencontrées, nous en avons rencontrés quelques-unes mais aucune entrainant un blocage dans l'avancée du projet. Les principales étaient, d'abord la plus longue, la lecture et compréhension du code fournis, un classique des langages assembleur. Il a fallu un long temps de travail préliminaires avant de pouvoir commencer à programmer. La seconde était l'optimisations du nombre de registres utilisés. À ceci s'ajoute d'autres soucis moindres : l'indexage des tableaux exprimés en octets, l'inversion des axes du display, le choix des structures et le grand nombre d'opérateurs que l'émulateur Mars fournis (lequel choisir).

Nous avons dès le départ su que les fonctions utiliseraient beaucoup de conditionnel avec différentes sorties possibles dépendant de ce qu'il y aurait en entrée. Nous nous sommes donc tout de suite orientées vers le *switch.* Avec l'aide de l'exemple ci-contre trouvé sur internet, nous avons modelé un *switch* à notre goût afin d'avoir un code sous une forme la plus lisible (pratique pour le débogage) et compréhensible pour nous et autres personnes pouvant lire le code par après. D'où la forme de notre code. Il en convient que le nombres de labels est plus élevées qu'une forme classique or de la rigueur et une convention de nommage pallie au problème. L'utilisation de boucle *while* était aussi multiple mais aucunes explications sur ce format à exprimer car classique dans ce langage.

```
$t1 = var
$t2 = i
$t3 = 4
$t4 = 1

lw $t1,0($i)
lw $t2, 0($var)

li $t3, 4                               #load intermediate value of 4 into $t3
li $t4, 1                               #load intermediate value of 1 into #t4
lui $t5, 0xFFFF
li $t6, 0x2000A0000

bltz $t1, DEFAULT_BODY #branch to default if var less than 0
slt $s0, $t1, $t3                   # if t1 < t3, then $s0 = 1, otherwise 0
beq $s0, $t4, DEFAULT_BODY  #if $s0=1 then branch to default
j C0_COND


C0_COND: addi $t1, $zero, 0   # case 0: set var to 0
            beq $t1, $t1, C0_BODY
                    j C1_COND

C1_COND: addi $t1, $zero, 1
                    beq $t1, $t1, C1_BODY
           j C2_COND

C2_COND: addi $t1, $zero, 2
                    beq $t1, $t1, C2_BODY
           j END

C0_BODY: sub $t2, $t2, $t4
                    j END
C1_BODY: addi $t2, $t2, 5
                    j END
DEFAULT_BODY: addi $t1, $zero, $t5
              j END
END: sw $t7, $t6
```

Pour les structure de données (dans le *.data*) de bases (serpent, obstacles, bonbon) sont restés les mêmes. À propos des ajouts, il y en a des simples : une phrase qui accompagne le score dans le terminal en fin de partie (phraseFinGame), le temps de sommeil (milli) pour qu'il soit modifiable afin de définir le système de niveaux… et d'autres plus complexes : pour le Rainbow Snake nous avons choisi un tableau qui boucle pour la simplicité et l'esthétisme. L'affichage du score graphique était le plus gros travail, nous avons choisi de déclaré deux tableaux par chiffres, un pour la position en X et un autre pour la position Y. Tous les chiffres tenant dans un rectangle de 5 par 7 (*Figure 1)*, nous avons préférer écrire les coordonnées dans ce format là puis centrer les chiffres avec un décalage par addition (*Figure 3*). Une boucle affiche pixels par pixels les chiffres or chaque caractère n'ayant pas le même nombre de pixels il fallut stocker (en première position) de chaque tableau X le nombre de pixels. Dans les tableaux Y un 0 est en première position pour que l'indice d'un pixel dans le tableau Y soit le même que dans le tableau X (*Figure 2)*.

Figure 1 Chiffres de 0 à 9

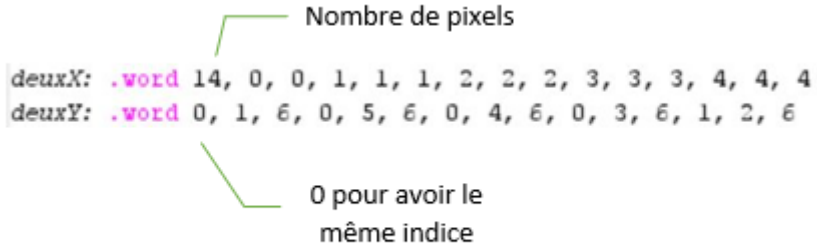

Figure 2 Déclaration du chiffre 2

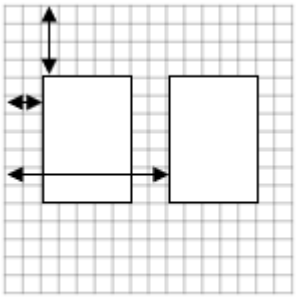

Figure 3 Décalage pour centrer