UFR de mathématique et d'informatique
Université de Strasbourg

PROJET SNAKE

Mips Assembly

Auteurs :

KARDAVA Elene
KOUSKOUSSI Aymen

1 Partie Projet

1.1 MajDirection

```
################################ majDirection ################################
# Param?tres: $a0 La nouvelle position demand?e par l'utilisateur. La valeur
#                 ?tant le retour de la fonction getInputVal.
# Retour: Aucun
# Effet de bord: La direction du serpent ? ?t? mise ? jour.
# Post-condition: La valeur du serpent reste intacte si une commande ill?gale
#                 est demand?e, i.e. le serpent ne peut pas faire de demi-tour
#                 en un unique tour de jeu. Cela s'apparente ? du cannibalisme
#                 et ? ?t? proscrit par la loi dans les soci?t?s reptiliennes.
##############################################################################
```

But : Met la direction du serpent à jour.

Axiome : si 4 (aucune entrée/ autre entrée) alors la même direction qu'avant. Sinon 0 - haut 1- droite 2-bas 3-gauche, 0 - bas.

On prend l'entrée de l'utilisateur $a0 et nous la stockons en dans le registre temporaire $t1. Nous comparons l'entrée avec la direction du snake. La fonction **autreDirection** est appelée chaque fois que la valeur de la direction et la valeur d'entrée ne sont pas égales. Il reprend la direction et vérifie la valeur pour d'autres fonctions. Lorsque la comparaison est faite, nous passons automatiquement à la fonction **finMajDirection** qui saute à l'adresse de retour.

1.2 UpdateGameStatus

```
############################## updateGameStatus ##############################
# Param?tres: Aucun
# Retour: Aucun
# Effet de bord: L'?tat du jeu est mis ? jour d'un pas de temps. Il faut donc :
#                 - Faire bouger le serpent
#                 - Tester si le serpent ? manger le bonbon
#                   - Si oui d?placer le bonbon et ajouter un nouvel obstacle
##############################################################################
```

But : Déplace le serpent, teste si le serpent a mangé le bonbon. Si oui , on augmente la taille et on crée un nouvel obstacle.

On alloue les places dans la pile pour stocker les variables.($s0-$s3) et nous stockons les coordonnées x et y du serpent dans des registres temporaires. Dans la fonction **siCandy**, on prend la position (x, y) du Candy et on compare avec le (x, y) de snake. S'ils ne sont pas égaux, le serpent continue à se déplacer et nous sautons à la fonction **decalage**.

- **Cas 1 :**

  S'ils sont égaux (c'est-à-direque le serpent a mangé le bonbon) alors on saute à la fonction **AugmTaille** qui prend la taille du snake, l'incrémente de 1 et remplace la

valeur de l'ancienne taille par la nouvelle. L'étape suivante consiste à générer une nouvelle position pour le bonbon dans la fonction **NouvPositionCandy** pour laquelle on appelle la fonction **newRandomObjectPosition** (définie dans le code initial). On remplace la position précédente du bonbon par la nouvelle générée. La fonction **NouvObstacle** adresse de tableaux des coordonnes des obstacles, appelle la fonction **newRandomObjectPosition** et remplace la position précédente d'obstacle. Les fonctions **AugmScore** et **AugmObstacle** incrementent le score et le nombre d'obstacles par 1.

- **Cas 2 :**

  S'ils sont égaux (c'est a dire le serpent n'a pas encore mange le bonbon), il continue à se déplacer.

  Axiome : si la taille de snake est égale à 1, alors on déplace seulement la tete.
  On appelle la fonction decalageTete, qui prend en entrée la direction et par conséquent appelle les fonctions *decalageTeteHaut, decalageTeteDroite, decalageTeteGauche, decalageTeteBas etc.*

  decalageTeteGauche : chaque fois que le serpent se déplace vers la gauche, la valeur de y est diminuée par 1 (x et y est inverse dans notre grille) ;

  decalageTeteHaut : : chaque fois que le serpent se déplace vers le haut, la valeur de x est incrémentée par 1 (x et y est inverse dans notre grille) ;

  decalageTeteDroite : chaque fois que le serpent se déplace vers la droite, la valeur de y est incrémentée par 1 (x et y est inverse dans notre grille) ;

  decalageTeteBas : chaque fois que le serpent se déplace vers le bas, la valeur de x est diminuée par 1 (x et y est inverse dans notre grille) ;

  Chaque fois que le serpent bouge, les directions sont mises à jour dans la pile par la fonction findecalage.

  Si la taille du serpent est supérieure à 1, alors on déplace le corps. Quand la queue du serpent prend la position de la tête la fonction **loop** met x-1 à la position de x et y-1 a la position de y (c'est-à-dire que l'on avance le corps), ensuite on appelle la fonction **decalage** qui effectue la mise à jour de la tête du serpent.

1.3 conditionFinJeu

```
############################## conditionFinJeu ##############################
# Param?tres: Aucun
# Retour: $v0 La valeur 0 si le jeu doit continuer ou toute autre valeur sinon.
#############################################################################
```

But : Nous observons si une des conditions de mort du serpent a été rencontrée. Retourne $v0 = 0 si aucune des conditions de fin de jeu a été rencontrée ou 1 si l'une d'entre elles l'a été. Si la valeur n'est pas égale à 0 , la boucle main va être executée 447 'bnez $v0 gameOver'. On jump à gameOver qui nous envoie à affichageFinJeu qui se chargera de l'affichage du score final.

On teste respectivement si la position du serpent est la même que :

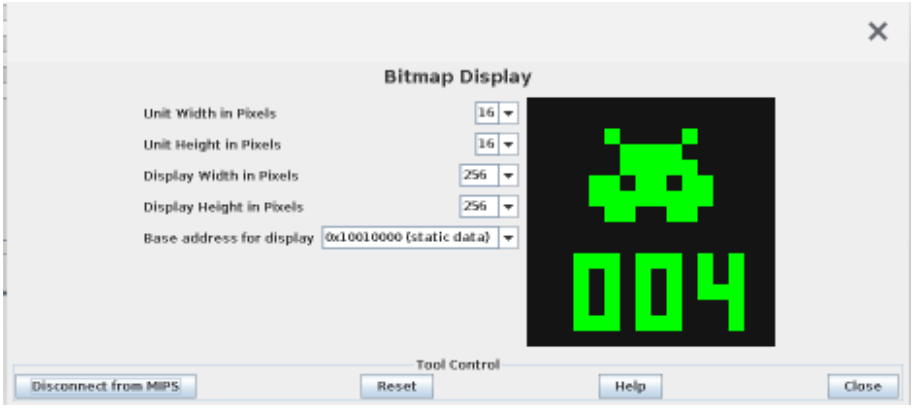
1. Les obstacles : On compare les coordonnées du serpent aux obstacles du premier jusqu'au dernier généré.
2. Une des cases du serpent (Cannibalisme) : On regarde dans TailleSnake la taille du serpent puis on verifie si les coordonnées de l'une des cases du serpent est la même que celui de la tête du serpent.
3. La bordure : on verifie si la position de la tête du serpent est la même que les bordures de la grille.

1.4 affichageFinJeu

```
############################## affichageFinJeu ##############################
# Param?tres: Aucun
# Retour: Aucun
# Effet de bord: Affiche le score du joueur dans le terminal suivi d'un petit
#                mot gentil (Exemple : ?Quelle pitoyable prestation !?).
# Bonus: Afficher le score en surimpression du jeu.
#############################################################################
```

But : Afficher le score final en message puis graphiquement.

On affiche le score en texte, puis nous appelons la fonction **resetAffichage** afin de remettre toutes les cases de la grille à notre couleur de fond. Afin de faire l'affichage graphique , on prend le score et le divise par 100 afin de l'avoir en trois chiffres(dans le format 010 pour un score de 10 par exemple). On passe ensuite la valeur de la centaine à $s0 à l'aide de 'mflo $s0 '. On compare la valeur de la centaine à zéro stockée dans $s0 , 99,9%. elle sera égale à zéro car le score de 100 est rarement dépassé. Le "beqz $s0 CentaineZero" nous enverra alors à la fonction chargée de prendre en adresse les coordonnées x et y de Numero0 chargé en .data . Le fonctionnement est le même pour la dizaine et l'unité(On utilise mfhi pour prendre le reste de la division). On affiche ensuite notre monstre Rétro dans les coordonnées que nous avons spécifié dans .data. Nous avons cherché une image d'un monstre rétro que l'on pourrait dessiner sur une grille aussi petite. Nous avons ensuite utilisé un outil en ligne **pixilart.com** pour le dessiner et en extraire les coordonnées.



1.5 Rainbow

But : afficher le serpent aux couleurs de l'arc-en-ciel

Avec les couleurs déjà mentionnées dans le code initial, on crée une table de couleurs appelée **Rainbow** et ajoute 7 couleurs arc-en-ciel plusieurs dizaines de fois. Puis, on donne à $s0 la valeur de 0 et dans la fonction **printSnake** qui affiche le serpent, on stocke Rainbow dans $a0. Donc en fonction de **PSloop**, à chaque fois qu'il affiche une partie de serpent, il prend la $s0-ième couleur du Rainbow et incrémente le valeur de $s0 (boucle $s0++).

1.6 Petite touche personnelle

Nous étions exaspérés de devoir à chaque fois passer de azerty (pour Aymen) à qwerty (pour Elene) et vice-versa , nous avons donc ajouté les touches de qwerty pour les mouvements du snake.

2 Difficultés rencontrées

- Nous avons perdu beaucoup de temps au début du projet à cause de l'inversion de l'axe des abscisses et des ordonnées. En initialisant la coordonnée y à 8 par exemple , elle allait sur l'axe des abscisses au lieu d'aller sur l'axe des ordonnées et vice-versa et nous pensions qu'il y avait un problème notre code.

- Nous avons également rencontré beaucoup de soucis avec les variables temporaires . Le fait qu'elles n'aient pas de nom spécifique rendait la chose difficile à suivre et rallongeait notre temps de programmation. Nous avons dû écrire sur un bloc-notes à chaque étape d'exécution à quelle valeur correspondait la variable analysée. Ceci nous a notamment posé problème pour accéder à une case du tableau des obstacles. Pour le tableau du serpent nous avions utilisé une variable temporaire déjà utilisée dans le tableau des obstacles Random.

- Nous avons eu un souci qui nous avait gravement ralenti et qui était l'apparition d'un obstacle en (0,0) à chaque fois que le serpent mangeait une deuxième pastille. C'était parce que l'on affichait l'obstacle avant la génération des coordonnées. Par défaut , l'obstacle sera placé dans la position initiale (0,0).

- Pour l'affichage du score graphique , nous avions commencé par afficher les chiffres pixel par pixels et ça nous avait pris des centaines de lignes dans le code. Nous avons ensuite eu l'idée de faire une liste pour les coordonnés x d'un chiffre et les coordonnés y du même chiffre.

3 GamePlay