Université de Strasbourg
UFR de mathématique et d'informatique

L2, Semestre 3, Automne 2021
BEAUREPERE Mathias, GALL Léon

Architecture des ordinateurs

Projet : Jeu du Snake

Le projet d'architecture des ordinateurs avait pour sujet l'implémentation du célèbre jeu snake en assembleur MIPS.

1 Choix d'implémentations

1.1 Style

Nous avons choisi de reprendre le style de programmation de la partie du projet fournie en amont par les professeurs. C'est pourquoi nous avons mis des indentations uniquement pour les boucles, mais nous avons préféré utiliser des virgules pour séparer les registres dans les instructions, par souci de clarté.

Nous nous sommes conformés aux normes MIPS, en utilisant les registres sauvegardés dans nos fonctions appelant d'autres fonctions en interne (en les sauvegardant en amont dans le prologue puis en les restaurant dans l'épilogue). Dans les fonctions n'appelant pas de fonctions ou pour des informations non utilisées après l'appel d'une fonction, nous avons utilisé les registres temporaires.

1.2 Fonctions

Nous avons eu à implémenter les fonctions suivantes.

Mise à jour de la direction

La fonction majDirection utilise la direction donnée par getInputVal, puis vérifie que le serpent ne fait pas de demi-tour à l'aide de la formule suivante, qui économise de nombreuses conditionnelles.

$$direction \neq (direction + 2) \mod 4$$

Si il ne fait pas demi-tour, la direction du serpent est mise à jour.

Évolution du jeu

La fonction updateGameStatus fait avancer le serpent, en décalant ses positions d'une case dans son tableau de positions et en modifiant l'emplacement de la tête. Ensuite, elle compare la position de la tête du serpent avec celle du bonbon. Dans le cas où elles sont à la même position, le serpent grandit d'une case (ceci est rendu possible par le fait que lors de l'avancement du serpent, sa dernière position est également déplacée, malgré le fait qu'elle ne soit plus lue. Pour la lire à nouveau, il suffit simplement d'incrémenter la taille du serpent). Un nouvel emplacement aléatoire pour le bonbon est alors trouvé et un obstacle (ou deux suivant le niveau) est également ajouté.

Conditions de fin du jeu

La fonction conditionFinJeu vérifie après chaque évolution du jeu que le serpent ne rentre ni dans lui-même, ni dans un obstacle ou encore que le serpent ne sorte pas de la grille. Pour ceci, on compare la position de la tête avec la queue du serpent, avec chaque obstacle (depuis le tableau d'obstacles), ainsi qu'avec la position des bords.

Affichage de fin du jeu

La fonction affichageFinJeu affiche graphiquement le score du joueur et le réécrit suivi d'un message de consolation dans la ligne de commande.

1.3 Fonctionnalités ajoutées

Affichage graphique du score

Pour structurer nos données, nous avons créé un tableau statique par chiffre, contenant les pixels de celui-ci. Il est agencé de la façon suivante : la première case de ce tableau contient sa taille, et la suite est composée de la position en ordonnée suivie de la position en abscisse de chaque pixel. Un autre tableau statique contient les labels de chaque chiffre, afin qu'en accédant à la bonne case de ce tableau, on puisse accéder à l'adresse du tableau de pixels du chiffre

correspondant. Cette structure de données permet l'économie d'un nombre important de conditionnelles, au prix d'une lisibilité légèrement réduite.

On réinitialise le dernier affichage graphique. Ensuite, nous regardons si le score est un chiffre ou un nombre et s'il s'agit d'un chiffre nous l'affichons au centre, et sinon nous récupérons le chiffre des dizaines et des unités pour les afficher l'un à côté de l'autre de façon centrée. Le score sera toujours inférieur à 100 (donc composé au maximum de deux chiffres), puisqu'il est techniquement pas possible de dépasser le score 86 en suivant notre système d'évolution, du fait du nombre limité de pixels. Pour afficher graphiquement un chiffre, on accède à son tableau de pixels et on affiche successivement chaque pixel en rouge.

Système de niveau

Nous avons également mis en place un système de niveau, qui évolue suivant le score. Chaque fois que le score dépasse un multiple de cinq, le niveau augmente. Par exemple, le niveau 0 est composé des scores 0 à 4 (inclus).

Pour que la difficulté augmente avec le niveau, nous avons fait en sorte d'ajouter deux obstacles chaque fois qu'un bonbon est mangé, dès le niveau 1. De plus, la vitesse du jeu augmente également de la façon suivante : Le temps d'attente entre deux évolutions est par défaut 500 millisecondes, et à chaque niveau, ce temps d'attente diminue suivant la formule :

$$sleep = 500 - 50 \times niveau$$

Si le niveau dépasse 8 (ce qui est extrêmement peu probable), ce temps cesse de diminuer et se comporte comme si le niveau était égal à 8.

Dès que le niveau 3 est atteint, le serpent s'affiche en mode arc-en-ciel.

Rainbow snake

Nous avons créé un tableau statique qui est rempli à partir du niveau 3 suivant la taille du serpent, par des couleurs aléatoires dont le code hexadécimal est compris entre 0x222222 et 0xEEEEEE, afin d'obtenir une couleur ni trop sombre, ni trop claire. Lors du tirage d'une couleur, on vérifie qu'elle n'est pas trop similaire à la couleur d'une autre partie du serpent. Lors de l'affichage, les couleurs de chaque partie du serpent sont récupérées dans ce tableau, et la couleur de la tête reste inchangée afin de ne pas perturber le joueur.

2 Répartition du travail et difficultés rencontrées

2.1 Répartition du travail

Nous avons choisi de nous répartir les fonctions, afin de pouvoir travailler en parallèle, chacun sur ses fonctions en utilisant GitLab pour se partager nos versions. Avant chaque fonction, nous nous sommes mis d'accord sur ce qu'il fallait faire et comment nous allions le faire. Donc la conception des fonctions s'est faite à deux, mais leur implantation s'est faite chacun de son côté. Bien évidemment, nous avons débuggé nos fonctions ensemble lorsque nous avions des problèmes que nous n'arrivions pas à résoudre seuls.

Mathias a programmé les fonctions conditionFinDeJeu, affichageFinJeu et de la fonctionnalité arc-en-ciel tandis que Léon a implémenté les fonctions majDirection, updateGameStatus, d'affichage graphique et du niveau.

2.2 Difficultés rencontrées

Nous n'avons pas rencontré de difficultés particulières lors de la réalisation de ce projet. Programmer en assembleur demande de la rigueur, et nous avons apprécié le défi imposé. Pour palier à la difficulté de relecture du code assembleur, nous avons commenté les instructions, ce qui nous a permi de comprendre le code de l'autre, et notre code à nous aussi parfois...

3 Observations

Nous avons observé que l'axe des abscisses et celui des ordonnées étaient inversées. Il nous a donc fallu adapter nos fonctions et structures de données en conséquence. Mais cela n'a en aucun cas perturbé notre réalisation.

Nous avons apprécié ce projet qui fût très enrichissant. Il nous a entre autres permis d'élargir notre connaissance du langage MIPS.