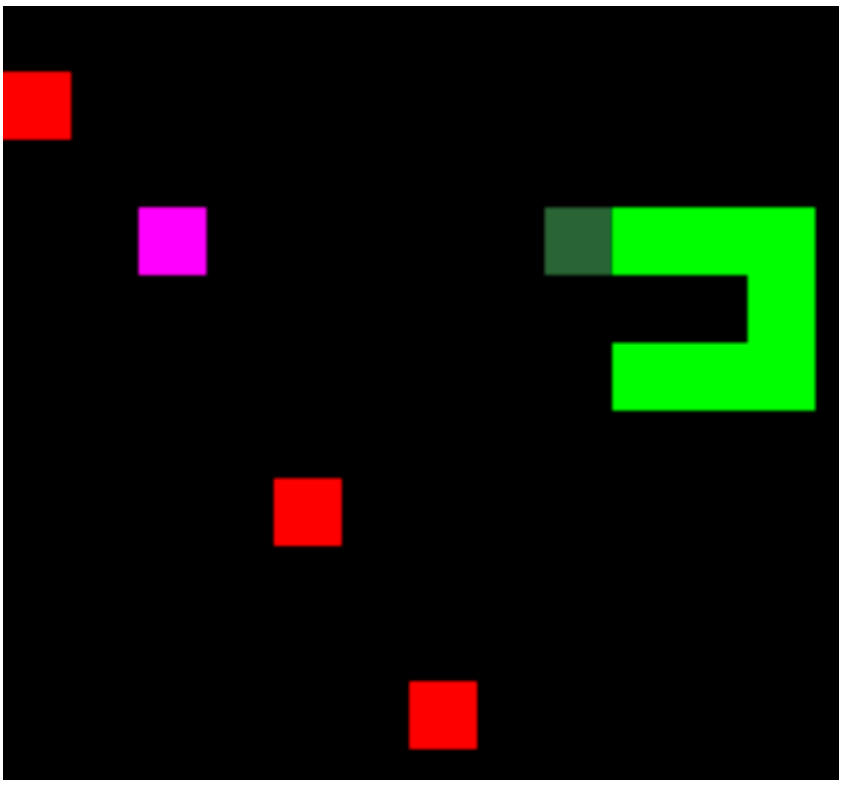
JEU DU SNAKE

SOMMAIRE



Introduction

Ce projet a comme objectif de créer le fameux Jeu Du Snake en utilisant un langage de bas niveau, l'assembleur MIPS. Notre objectif est d'utiliser le code fournis et de le complété en construisant quatre fonctions essentielles pour avoir un jeu qui fonctionne a 100%. Dans ce rapport nous vous présentons nos explications et méthodes de conception du code des différents fonctions et les difficultés rencontrées pendant le projet. Nous allons commencer par la fonction *majDirection*, puis la fonction *updateGameStatus*, ensuite la fonction *conditionFinJeu* et enfin la fonction *affichageFinJeu*.

I) majDirection

conception :Karim - Hussein
code : Karim

Cette première fonction sert a filtrer les erreurs possibles durant le mouvement du snake dans la grille. Dans notre cas on défini une erreur de mouvement par une demande de l'utilisateur de changement de direction du snake a une direction opposée a celle que le snake avait, et qui dans ce cas ne prend pas en compte la commande donner au snake et continue le même trajet qu'il avait.
Pour effectuer cela, on a commencer par utiliser le registre $t0 dans lequel on a placé snakeDir qui détermine la direction du movement du snake et on va le comparer a la valeur de du registre $a0 qui contient la valeur de direction souhaitée par l'utilisateur entrer au clavier. La fonction va donc comparer la valeur de snakeDir avec tous les valeurs de directions possibles, ici (Up=0, Right=1, Down=2, Left= 3) et si snakeDir vaut une des 4 valeurs possibles, il compare ensuite la valeur de snakeDir avec la valeur que l'utilisateur va entrer sur son clavier qui est dans le registres $a0. Si la valeur de $a0 est égale ou opposé a la valeur de snakeDir alors on fait un jump vers le label "exit" qui permet de sortir de la fonction et qui continue a exécuter les fonctions qui suit, sans modifier le output (direction du snake). Sinon la fonction fait un jump vers le label "store", snakeDir prend la valeur de $a0 et la stock dans son adresse memoir.

II) updateGameStatus

conception : Hussein - Karim
code : Hussein

Cette deuxième fonction est la plus charger entre les quatre fonctions. Son rôle et de faire bouger le snake, vérifier si le snake a manger un bonbon et si oui la taille du snake doit s'agrandir et enfin on génère un nouveau obstacle et un nouveau bonbon lorsque le snake mange un bonbon.
On charge dans $t0 la valeur de tailleSnake avant la sous fonction *moveBx* ce qui nous permet de prendre la valeur et la modifier dans la fonction sans reprendre la valeur original avec chaque iteration. La sous fonction *moveBX* sert a faire bouger le corps du snake avec la tete et non pas isoler que le mouvement de la tete. Donc le but de cette sous fonction et faire un shift de tous les valeur dans la table de snakePosX et snakePosY vers la droite, en faisant cela le corps et aussi en mouvement dépendant de la tete. Sans cette fonction, quand le snake mange un bonbon la nouvelle partie du corps ce trouve au coordonnées de depart qui sont (0,0), ce qui nous a poser une difficulté pour un certain temps. Pour realiser le shift ont a besoin de commencer de la dernière case de la table pour traverser toute la table et en même temps déplacer toutes les valeurs de chaque case a la case d'après. Grace a l'équation (snake_PosX/Y + (tailleSnake *4)) on peut traverser la table de la dernière case vers la première. Dans la sous fonction if_ate_candy on compare les coordonnées de Candy et du snake. Candy contient le coordonnées X du bonbon et Candy + 4 le coordonnées Y. Donc on compare ces deux coordonnées avec les coordonnées de la tete du snake (snakePosX - PosY). Si les coordonnées sont égaux on fait un jump vers ate_condition qui permet d'agrandir le snake, générer un nouveaux bonbon et obstacle, incrémenté le score par 1, et enfin accélére le mouvement du snake.

Pour agrandir le snake il suffit d'incrémenter tailleSnake par un, ce qui agrandit la table de PosX et PosY du snake, il faut également faire cela a numObstacles car on génère un nouveau obstacle chaque fois le snake mange un Candy. Pour générer un nouveau bonbon/obstacles il suffit de faire un jump and link vers *newRandomObjectPosition* qui nous renvoie des coordonnées X($v0) et Y($v1) qu'ont va ensuite les stocker dans les coordonnées du bonbon et de l'obstacles. Une des difficultés qu'on a rencontré est le stockage des nouveaux coordonnées des obstacles dans les cases qui suit directement les derniers coordonnées du dernier obstacle mis en place. Sans ceci l'obstacle apparaît sur la grille mais n'arrête pas le jeu quand le snake le touche.

III) conditionFinJeu

conception : Karim - Hussein
code : Karim

La troisième fonction vérifie trois critère, si le snake dépasse pas les bords de la grille (15,15 : 0,0), si le snake et on contacte avec un obstacle, enfin si le snake est en contact avec son corps.
On compare les coordonnées de la tete du snake si une des deux coordonnées dépasse 15 ou est inférieur a 0 on arrête le jeu.
La sous_fonction "incontact_with_ob" et "increment_ob" sont deux partie d'une même sous_fonction. On compare les coordonnées de la tête avec ceux des obstacles.
Si les comparaison sont invalides on signal une erreur, sinon on avance dans notre mémoire de 4 octet pour ensuite comparer les coordonnées d'un different obstacle dans la grille.
On répète le même processus des deux sous fonctions "incontact_with_ob" et "increment_ob" pour les sous fonction "contact_with_snake" et "increment_sXY" mais au lieu de comparer les coordonnées de la tete avec ceux des obstacles on les compares avec son corps luis même. Donc logiquement on commencera a partir de la première valeur du corps dans le tableaux et non pas la tête.

IV) affichageFinJeu

code: Hussein

La quatrième fonction est simple, elle affiche le score du jeu suivît d'un message de choix.
Pour afficher un string de caractère il suffit de initialiser la valeur 4 dans le registre $v0, puis on charge dans l'adresse (la) de $a0 le label de notre message "GameOver", puis le "syscall" affiche tous cela.
Même chose pour le scoreJeu mais cette fois on initialise la valeur 1 dans $v0 car on veut afficher un int et non pas un message, puis on utilise "lw" car on charge une valeur de l'adress et non pas l'adresse lui meme dans $a0, puis le "syscall" affiche tous cela.
Meme chose pour le message "tryagain".