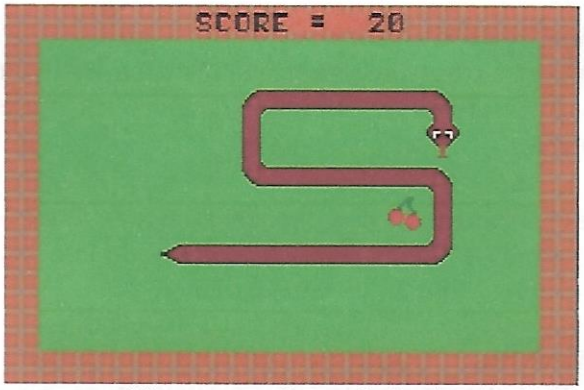
Projet B23
Développement d'un jeu de Snake

Enseignant : Hazem Wannous

1. Introduction

Ce projet vise le développement d'une variante du jeu populaire connu sous le nom de Snake (serpent en français), qui est un jeu vidéo créé au milieu des années 1970.
Le joueur contrôle une longue et fine créature semblable à un serpent, qui doit slalomer entre les bords de l'écran et les obstacles qui parsèment le niveau. Pour gagner chacun des niveaux, le joueur doit faire manger à son serpent un certain nombre de fruits, allongeant à chaque fois la taille de celui-ci.



Une version de jeu de Snake

2. Descriptif du projet

Vous devez écrire un programme C permettant de jouer au Snake en différentes façons et à plusieurs niveaux en mode texte.

a) Déplacement manuel dans un labyrinthe

Un labyrinthe est une pièce découpée en cases qui sont soit libres, soit des obstacles, soit des murs. Le programme commence par charger un labyrinthe, et place le serpent sur une case de départ. Le joueur se déplace ensuite dans une direction (haut, bas, droite, gauche) et marque des points en mangeant des fruits placés initialement dans un labyrinthe.

Le déplacement du serpent est réalisé manuellement par le joueur à l'aide des touches de clavier qui orientent la tête du serpent et son corps suit. La queue du serpent se développe par un segment à chaque fois que le serpent mange un fruit.
Le jeu se termine quand le serpent heurte le mure, lui-même, ou le temps expire. Le score final est calculé en fonction de la longueur de sa queue.
Option : le Labyrinthe peut contenir aussi des obstacles mortels déjà présentes ou qui peuvent apparaître soudainement, le joueur devra les éviter.

b) Serpent en mouvement:

La partie commence par charger un plateau – rectangle de largeur et hauteur définies dont le serpent ne peut s'échapper – et place le serpent sur une case de plateau mais en mouvement dans une direction initiale définie. La direction de la tête du serpent peut être tournée horizontalement et verticalement à l'aide des touches de clavier (haut, bas, gauche, droite) et le corps du serpent suit la tête. La queue se développe par un segment à chaque fois que le serpent mange un fruit. Le joueur marque des points en mangeant des fruits placés aléatoirement sur le plateau.

Il y a plusieurs niveaux de jeu (trois), chacun avec une difficulté croissante. Le serpent est toujours en mouvement et se déplace plus vite à chaque niveau. Lorsque le joueur collecte suffisamment de fruits, il progresse sur le niveau suivant, où le serpent devient plus long, et la quantité de fruits à collecter pour progresser dans le niveau devient plus grande.

Le jeu se termine quand le serpent se heurte à un obstacle mortel, le mure, ou lui-même. Un score est obtenu à la fin de chaque partie et sauvegardé, qui est calculé en fonction de la longueur du serpent et le temps consumé.

c) Serpent en mouvement à travers les murs

Seule différence par rapport au cas précédent « serpent en mouvement » est que le joueur possède plusieurs vies et le serpent se déplace à travers les mures à chaque fois qu'une nouvelle vie se met en place.

Le but du jeu dans toutes les parties est de collecter les points (fruits) et d'éviter les obstacles, les mures et le serpent lui-même. Quand vous recueilliez de la nourriture, le serpent devient plus long, augmentant ainsi vos chances de vous percuter.

d) Joueur vs ordinateur

Proposer une méthode d'intelligente artificielle permettant de gérer l'évolution du serpent dans son environnement, afin qu'un joueur peut jouer contre l'ordinateur

Deux aspects devraient être pris en considération lors de la conception d'un algorithme (solution automatique): (1) La sûreté: l'algorithme devrait garantir la vie de serpent pendant une longue période. Ce qui signifie que chaque étape doit être sûr non seulement temporellement, mais aussi pour une longue course. (2) La concurrence: l'algorithme devrait également être en mesure de proposer une solution plus compétitive face à joueur.

Deux stratégies peuvent être considérées:

- *Shortest path*: la meilleure solution est le chemin le plus court au fruit, car il est peu probable que ces mouvements vont conduire le serpent dans une situation dangereuse.
- *Safe Path*: parfois la stratégie du plus court chemin serait extrêmement dangereuse, alors il faut éviter chemin.

3. Les règles sur des éléments du jeu

Le **serpent** possède une queue de taille variable mais qui ne peut que grandir. A chaque mouvement, le serpent choisit l'une des quatre directions cardinales et avance d'une case dans cette direction, c'est-à-dire que sa tête arrive sur la case en question et que la case sur laquelle se trouvait le bout de sa queue devient vide. Les autres parties de son corps ne bougent pas.

Les **mouvements** du serpent sont limités aux déplacements d'une case à une autre et le serpent ne peut pas rester entre deux cases.

L'environnement de jeu dans lequel le serpent se déplace est un **plateau** – rectangle de largeur et hauteur définies – dont le serpent ne peut s'échapper.
Les cibles (**fruits**) apparaissent de façon aléatoire sur le plateau, mais un fruit ne peut apparaitre ni sur les murs ni sur un partie du serpent. Lorsque le serpent mange un fruit, la taille du serpent augmente au coup suivant (la tête avance mais le bout de la queue n'est pas retiré).
En plus des mures, des **obstacles** supplémentaires peuvent être rajoutés sur le plateau, ce qui rend le mouvement de serpent plus contraignant (optionnel).

En fin, le corps du serpent est également considéré comme un obstacle, et la partie s'arrête lorsque le serpent entre en collision avec un morceau de sa queue ou avec un mur. Il faut donc prévoir une structure conservant les positions de tous les points de la queue du serpent tout au long de la partie.

Il est important que vous réfléchissiez soigneusement à la décomposition de votre programme en termes de sous-programmes (fonctions et procédures). De nombreux sous-programmes doivent impérativement être identifiés (chargement du niveau, déplacement, affichage du résultat, ...). Attention à bien séparer le code qui gère l'affichage du code qui réalise le noyau fonctionnel
Réfléchissez aussi aux paramètres attendus et résultats fournis par ces sous-programmes. Le programme principal doit être une combinaison d'appels aux différents sous-programmes.
Voici quelques indications pour l'écriture du programme.

4. Représentation des données

Vous aurez besoin de représenter informatiquement le plateau (et éventuellement le labyrinthe). Une solution est d'utiliser une matrice d'une taille fixe, avec des 0 pour les fruits, des 1 pour les murs... Il vous faudra également représenter les positions de départ, d'arrivée, et du joueur. Il faut également tenir compte de la représentation des autres obstacles s'il y a lieu.

La structure du serpent doit pouvoir contenir toutes les informations permettant de le localiser sur le plateau (position de la tête), ainsi que celles relatives à son état (direction, fruits mangés et peut-être vie/mort).

Mouvement

A chaque mouvement possible, il faut l'appliquer et mettre TOUTES vos données à jour.

Conditions d'arrêt

Les meilleures choses ont une fin, et généralement le jeu se termine quand le serpent se heurte à un obstacle mortel, le mure, lui-même, ou le temps expire, mais cela peut varier selon la partie.

5. Cahiers des charges

Ces objectifs sont à résoudre DANS L'ORDRE ! Les fonctionnalités les plus avancées ne seront pas comptées dans la note si les fonctionnalités les plus simples ne sont pas correctement implémentées, avec un code propre et commenté !

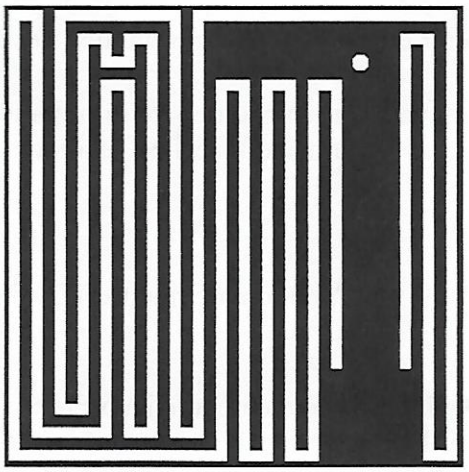
On n'exige que les fonctionnalités de base (a) et intermédiaire (b, c), le (d) permet de passer d'un bon à un excellent projet. Avant de passer d'un groupe de fonctionnalités au suivant, demandez au responsable de TP.

Base :
On doit pouvoir jouer une partie de Snake dans un labyrinthe avec une interface texte. En fin de partie, on a le choix de recommencer une partie ou de sortir du programme. Le jeu ne comporte qu'un niveau décrit "en dur" dans votre code.

Intermédiaire :
Cela correspond au développement des parties (b + c). Le jeu comprend maintenant plusieurs niveaux codés "en dur". On peut choisir son niveau en début de partie ou bien lorsque le joueur collecte suffisamment de fruits, il progresse sur le niveau suivant dans une stratégie de difficulté croissante (optionnel).
On compte le nombre de points nécessaires (fruits mangés) pour compléter le niveau, et on implémente une galerie des 3 meilleurs scores (une par niveau).

Avancé :
On cherche à implémenter une solution automatique (intelligente artificielle) afin qu'un joueur peut jouer contre l'ordinateur.
On stocke maintenant les niveaux dans des fichiers séparés, chargés au moment du choix du niveau. On propose une fonction de sauvegarde, qui permet de garder ses meilleurs scores, et sa position dans le niveau en cours.

Très avancé : (ATTENTION : LE RESTE DOIT ETRE CORRECT!)
Conception d'un algorithme qui permet de gagner n'import quelle partie de jeu avec un score parfait, ou encore le plus rapide possible.



Snake parfait

Eléments à remettre en fin de projet :
- Le code source de votre projet. Ce code source devra contenir les commentaires nécessaires à sa compréhension.