PROJET COWSAY

CAILLE Daniel / CRELEROT Thomas : IMA-4

Préliminaires

Arguments	Utilités	Exemples
cowsay (sans option)	Cowsay génère une image ASCII d'une vache disant quelque chose fourni par l'utilisateur	cowsay bonjour renvoie une vache disant bonjour avec le mot bonjour dans une bulle
-e eyestring	L'option -e permet de sélectionner l'apparence des yeux de la vache, auquel cas les deux premiers caractères de la chaîne d'argument eye_string seront utilisés pour les yeux de la vache , par défaut les yeux de la vache sont les caractères «oo».	cowsay -e xx 'texte' affiche une vache avec 'xx' à la place des yeux
-T tongue_string	L'option -T permet de sélectionner l'apparence de la langue de la vache, auquel cas les deux premiers caractères de la chaîne d'argument tongue_string seront utilisés pour la langue de la vache , par défaut il n'y a pas la vache n'a pas de langue visible	cowsay -T U 'texte' affiche une vache avec le caractère 'U ' représentant la langue
-f cowfile	L'option -f spécifie un fichier d'image de vache particulier à utiliser présent dans le répertoire des modèles de cowsay.	cowsay -f dragon-and-cow 'texte' affiche un dragon disant le texte entrer en argument et une petite vache sur le point de se faire attaquer
-h	cowsay -h est une commande qui affiche les différentes options présente sur la commande cowsay et montre l'usage de la commande cowsay	cowsay -h affiche ceci cow{say,think} version 303 (c) 1999 Tony Monroe Usage:cowsay [-bdgpstwy] [-h] [-e eyes] [-f cowfile] [-l] [-n] [-T tongue] [-W wrapcolumn] [message]

Arguments	Utilités	Exemples
-l	Cette option sert à afficher l'ensemble des fichier présents dans le répertoir de modèle qui peuvent servir de modèles pour l'option -f	cowsay -l affiche ceci Cow files in /usr/share/cowsay/cows: apt bud-frogs bunny calvin cheese cock cower daemon default dragon dragon-and-cow duck elephant elephant-in-snake eyes flaming-sheep fox ghostbusters gnu hellokitty kangaroo kiss koala kosh luke-koala mech-and-cow milk moofasa moose pony pony-smaller ren sheep skeleton snowman stegosaurus stimpy suse three-eyes turkey turtle tux unipony unipony-smaller vader vader-koala www
-n	l'option -n sert à faciliter l'utilisation de messages arbitraires avec des espaces arbitraires	cowsay -n 20 "Bonjour tout le monde !" la vache s'afficher et cela limitera le texte présent dans la bulle à 20 caractères par ligne
-W column	l'option -W spécifie approximativement où le message doit être encapsulé. La valeur par défaut est équivalente à -W 40, c'est-à-dire envelopper les mots au plus tard à la 40ème colonne	cowsay -W 2 abcd va afficher une vache avec une bulle ou il y aura écrit abcd avec une lettre par ligne
-b ou -d ou -g ou -p ou -s ou -t ou -w ou -y	Cette comande affiche une vache avec des yeux et/ou une langue différentes de d'habitude pour illsutrer un état émotionnel/physique particulier.	L'option -b lance le mode Borg ; -d donne une vache morte; -g invoque le mode gourmand ; -p provoque un état de paranoïa chez la vache ; -s donne à la vache une apparence complètement défoncée ; -t donne une vache fatiguée ; -w donne une vache excité; -t et initie le mode filaire ; -y apporte l'apparence juvénile de la vache.

Bash

numéro 1 : cow_kindergarten

Ce script affiche les nombres de 1 à 10 en utilisant cowsay

```
#!/bin/bash

clear
# Boucle pour compter de 1 à 10
for ((i=1 ; i<=10 ; i++))
do
    sleep 1 | cowsay $i # Utilise cowsay pour afficher le nombre actuel
    clear # Efface l'écran pour afficher le nombre suivant
done
cowsay -T " U" "c'est fini"
```

numéro 2: cow_primaryschool

Ce script affiche les nombres de 1 à n (un entier) avec cowsay.

```
#!/bin/bash


clear
# Boucle pour compter de 1 à n
for ((i=1 ; i<=$1 ; i++))
do
    sleep 1 | cowsay $i
    clear # Efface l'écran pour afficher le nombre suivant
done
cowsay -T " U" "c'est fini"
```

numéro 3 : cow_highschool

Ce script affiche les carrés des nombres de 1 à n avec cowsay.

```
#!/bin/bash


clear
# Boucle pour calculer les carrés des nombres de 1 à n
for (( i=1; i<=$1 ; i++ ))
do
    carre=$(($i**2)) # Calcule le carré de la valeur actuelle de la boucle
    sleep 1 | cowsay $carre  # Utilise cowsay pour afficher le carré actuel
    clear  # Efface l'écran pour afficher le carré suivant
done
cowsay -T " U" "c'est fini"
```

numéro 4 : cow_college

Ce script affiche les nombres tous les termes de la suite de Fibonacci inférieur à l'entier entrer en argument

```
#!/bin/bash


Un=0 # Initialisation du premier terme de la suite de Fibonacci
Un1=1 # Initialisation du deuxième terme de la suite de Fibonacci
clear

# Vérification si l'argument est inférieur ou égal à 0
if [ $1 -le 0 ]; then
    sleep 5 | cowsay "l'argument doit être strictement supérieur de 0 car il n'y a
pas de terme inférieur à 0 dans la suite de Fibonacci "
```

```
else
    sleep 1 | cowsay "0" # Affiche le premier terme "0" de la suite
    clear
    # Vérification si le deuxième terme est inférieur à l'argument
    if (($Un1 < $1)); then
        sleep 1 | cowsay "$Un1"
        clear
    fi
    # Boucle pour calculer les termes suivants de la suite de Fibonacci
    for ((i=2; i<=$1; i++))
    do
        Un2=$(($Un1+$Un)) # Calcul du terme suivant
        if (($Un2 < $1)); then
            sleep 1 | cowsay "$Un2" # Affiche le terme suivant s'il est inférieur
à l'argument
            clear
        else
            break # Sort de la boucle si le terme suivant est supérieur ou égal à
l'argument
        fi
        Un=$Un1 # Mise à jour des valeurs pour le calcul des termes suivants
        Un1=$Un2
    done
fi

clear
cowsay -T " U" "c'est fini"
```

numéro 5 : cow_university

Ce script affiche les nombres tous les nombres premiers inférieur à l'entier entrer en argument

```
#!/bin/bash


clear
# Boucle pour tester les nombres premiers jusqu'au nombre spécifié par
l'utilisateur
for ((i=1; i<$1 ; i++))
do
    compteur=0 # Initialisation du compteur de diviseurs
    # Boucle pour compter le nombre de diviseurs du nombre actuel
    for ((a=1; a<=$i ; a++))
    do
        if [ $(($i%$a)) -eq 0 ] # Test si $i est divisible par $a
        then
            compteur=$(expr $compteur + 1) # Incrémente le compteur de diviseurs
si c'est le cas

        fi
    done
```

```
    # Si le nombre de diviseurs est égal à 2, alors c'est un nombre premier
    if  [ $compteur -eq 2 ]
    then
        sleep 1 | cowsay $i # Affiche le nombre premier
        clear # Efface l'écran pour afficher le prochain nombre
    fi
done
cowsay -T " U" "c'est fini"
```

numéro 6 : smart_cow

Ce script prend en argument un calcul (par exemple:'3' + 11') entrer guillemet avec un espace entre chaque caractère et affiche une vache qui prononce le calcul et dont les yeux se transforment en le résultat du calcul

```
#!/bin/bash


# Sépare la chaîne en trois parties en utilisant l'espace comme délimiteur
part1=$(echo $1 | cut -d ' ' -f1)
part2=$(echo $1 | cut -d ' ' -f2)
part3=$(echo $1 | cut -d ' ' -f3)

# Vérifie si le deuxième élément est une opération
if [ $part2 = '+' ]
    then result=$(expr $part1 + $part3)  # Effectue l'addition
        if [ $result -lt 10 ]
        then cowsay -e $result" " $1 # Affiche le résultat avec un espace avant
s'il est inférieur à 10 pour ne pas créer de décalage
        else
            cowsay -e $result $1 # Sinon on affiche normalement
        fi

elif [ $part2 = '-' ]
    then result=$(expr $part1 - $part3) # Effectue la soustraction
        if [ $result -lt 10 ]
        then cowsay -e $result" " $1 # Affiche le résultat avec un espace avant
s'il est inférieur à 10 pour ne pas créer de décalage
        else
            cowsay -e $result $1 # Sinon on affiche normalement
        fi

elif [ $part2 = "x" ]
    then
        # Boucle pour effectuer la multiplication
        for (( i=0; i<$part3; i++ )); do
            result=$(expr $result + $part1)
        done
        if [ $result -lt 10 ]
        then cowsay -e $result" " $1 # Affiche le résultat avec un espace avant
s'il est inférieur à 10 pour ne pas créer de décalage
        else
```

```
            cowsay -e $result $1 # Sinon on affiche normalement
        fi


elif [ $part2 = '/' ]
    then result=$(expr $part1 / $part3) # Effectue la division
        if [ $result -lt 10 ]
        then cowsay -e $result" " $1 # Affiche le résultat avec un espace avant
s'il est inférieur à 10 pour ne pas créer de décalage
        else
            cowsay -e $result $1 # Sinon on affiche normalement
        fi

else
    # Affiche un message d'erreur si l'opération n'est pas reconnue et rappel
quelles sont les bon symboles et les conditions d'utilisation
    echo "Erreur: le bon symbole pour l'addition est '+' "
    echo "        le bon symbole pour la soustraction  est '-' "
    echo "        le bon symbole pour la division est '/' "
    echo "        le bon symbole pour la multiplication est 'x' "
    echo "        il faut un espace entre chaque caractère ( par exemple:'3' +
11') "

fi
```

numéro 7 : crazy_cow

Ce script calcule et affiche le nombre d'or avec un nombre (un entier) de décimales spécifié par l'utilisateur en argument .

```
#!/bin/bash


# Vérifie si le nombre d'arguments est différent de 1
if [ $# -ne 1 ]; then
    echo "il faut un seul argument, un entier positif."
    exit 1
fi

# Vérifie si l'argument est un nombre entier positif
if [ $1 -lt 0 ]; then
    echo "L'argument doit être un nombre entier positif."
    exit 1
fi

# Calcul du nombre d'or avec le nombre de décimales spécifié par l'utilisateur
nb_or=$(echo "scale=$1; ((1+(sqrt(5)))/2)" | bc -l) # Formule du nombre d'or =
((1+(sqrt(5)))/2)

cowsay "Le nombre d'or avec $1 décimale(s) est : $nb_or " # Affichage du nombre
d'or avec un message
```

```
# Affichage d'un message expliquant le lien entre le nombre d'or et la suite de
Fibonacci
cowthink "D'ailleur il y a un lien entre le nombre d'or et le suite de
Fibonacci vu précédement, car la limite de Fn/F(n-1) tend vers le nombre d'or
quand n tend vers plus l'infini"
exit 0
```

C

options.h

```
#ifndef OPTIONS_H
#define OPTIONS_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int options(int argc, char* argv[]);
void affiche_vache(char* texte,char* yeux, char* queue, char* langue , int
nb_vache);
void message(char *chaine);


#endif
```

new_cow.c

Après avoir utiliser make pour compiler tous les fichiers dans le bonne ordre, il faut entrer dans l'interpréteur de commande './newcow' suivi par des options ou non, avec un texte pour éxécuter ce programme.

Pour notre version de cowsay en C nommer newcow :

Nous avons mis l'option -e [caractères] pour pouvoir modifié les yeux de la vache (Seuleument les deux premier caractères s'affiche au niveau des yeux);

Nous avons mis l'option -T [caractères] pour pouvoir modifié la langue de la vache (Seuleument les deux premier caractères s'affiche au niveau de la langue);

Nous avons mis l'option --tail [entier] pour pouvoir modifié la taille de la queue de la vache de la vache ;

Il faut obligatoirement mettre un texte entre guillement à la fin pour que la vache s'affiche comme sur la vraie commande cowsay.

Exemple d'utilisation : ./newcow -e xx -T ' U' --tail 2 'je suis une vache'

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include "options.h"

// Fonction principale
int main(int argc, char* argv[]) { // Appel de la fonction pour gérer les options
    options(argc,argv);
    }
```

option.c

```c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "options.h"


// Fonction pour afficher le message dans la bulle
void message(char *chaine){
    int len = strlen(chaine);
    printf(" ");
    // Boucle pour réaliser la ligne supérieure
    for (int i = 0; i < len + 2; i++) {
        printf("_");
    }
    printf(" \n");
    // affiche le texte entouré par ces deux caractères < >
    printf("< %s >\n", chaine);
    printf(" ");
    // Boucle pour réaliser la ligne inférieure
    for (int i = 0; i < len + 2; i++) {
        printf("-");
    }
    printf(" \n");
}

// Fonction pour afficher une vache
void affiche_vache(char* texte,char* yeux, char* queue, char* langue , int
nb_vache) {
    message(texte); // Affiche le message dans la bulle avec le texte donné en
argument
    // Affiche la vache (par défaut le nombre de vache est 1 et il y a pas
d'option pour modifier cela donc on affiche toujours qu'une seule vache)
    for (int i=0;i<nb_vache;i++){
        printf("       \\    ^__^\n"
        "        \\   (%s)\\_______\n"
        "            (__)\\       )\\/\\%s\n"
        "             %s ||----w |\n"
        "                ||     ||\n", yeux, queue, langue);
    }
}
```

```c
// Fonction pour gérer les options
int options(int argc, char* argv[]) {
    char yeux[3] = "oo";  // Yeux par défaut
    char langue[3] = "  "; // Langue par défaut
    char* queue = NULL; // Initialisation de la queue
    int nb_vache=1; // Nombre de vaches (par défaut une seule) non modifiable
    char texte[1000]; // Message à afficher par la vache
    strcpy(texte, argv[argc - 1]); // Copie le dernier argument dans texte
    // Si seul le message est donné, affiche une seule vache avec les paramètres
par défaut
    if (argc==2){
          affiche_vache(texte,yeux, "", langue, nb_vache);
    }
    // Si des options sont données
    else if (argc %2 == 0 ){
        // Parcourt toutes les options
        for (int i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-e") == 0 || strcmp(argv[i], "--eyes") == 0) {
// Si l'option spécifie les yeux de la vache
                if (i + 1 < argc) {
                    strncpy(yeux, argv[i + 1], 2);
                    yeux[2] = '\0';
                    i++;
                } else {
                    printf("Erreur : argument manquant pour l'option -e\n");
                    return 1;
                }
            } else if (strcmp(argv[i], "-T") == 0) {  // Si l'option spécifie la
langue de la vache
                if (i + 1 < argc) {
                    strncpy(langue, argv[i + 1], 2);
                    langue[2] = '\0';
                    i++;
                } else {
                    printf("Erreur : argument manquant pour l'option -T\n");
                    return 1;
                }
            } else if (strcmp(argv[i], "--tail") == 0) { // Si l'option spécifie
la queue de la vache
                if (i + 1 < argc) {
                    int taille = atoi(argv[i + 1]);
                    if (taille >= 0 && taille <= 9) {
                        // Allocation dynamique de la queue de la vache
                        queue = malloc(taille * 3 + 1);
                        if (queue == NULL) {
                            printf("Erreur : échec de l'allocation de mémoire pour
la queue\n");
                            return 1;
                        }
                        // Construction de la queue de la vache avec un boucle en
fonction de l'entier entrer en argument apres le --tail
                        for (int j = 0; j < taille * 2; j += 2) {
                            queue[j] = '/';
                            queue[j + 1] = '\\';
```

```
                        }
                        queue[taille * 3] = '\0';
                        i++;
                    } else {
                        printf("Erreur : argument non compris entre 0 et 9 ou
manquant pour l'option --tail\n ");
                    }
                }
            }
        }
        // Affichage de la vache avec les options spécifiées
        if (queue != NULL) {
            affiche_vache(texte, yeux, queue, langue, nb_vache);
            free(queue);
        } else {
            affiche_vache(texte, yeux, "", langue, nb_vache);
        }
    }
    // Si le nombre d'arguments est incorrect
    else {
        printf("Le nombre d'arguments est incorrecte!\n");
        return -1;
    }
return 0;

}
```

wildcow.h

```
#ifndef WILDCOW_H
#define WILDCOW_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void wildcow(int argc,char *argv[]);


#endif
```

wildcow.c

Après avoir utiliser make pour compiler tous les fichiers dans le bonne ordre, il faut entrer dans l'interpréteur de commande './wildcow' suivi par des options ou non, avec un texte comme pour newcow car on s'appuye encore une fois sur les options et l'affichage présent dans le fichier option.c pour éxécuter ce programme

Ce programme affiche une série d'animations d'une vache à l'écran. Premièrement nous pouvons voir dans cette animation une vache avec des yeux clignotants, deuxièmement nous pouvons voir dans cette animation

une vache qui tire la langue plusieurs fois, et enfin nous pouvons voir dans cette animation une vache se déplaçant vers l'arrière puis vers l'avant en marchant (la vache se déplacant en arrière et en avant n'a pas d'option ni de texte).

Exemple d'utilisation : ./wildcow -e OO -T ' U' --tail 2 'je suis une vache'

```
#include<stdio.h>
#include <unistd.h>
#include "options.h"
#include "wildcow.h"

// Efface l'écran
void update() {
    printf("\033[H\033[J");
}

// Déplace le curseur à la position spécifiée
void gotoxy(int x, int y) {
    printf("\033[%d;%dH", x, y);
}

// Affiche une chaîne de caractères à la position spécifiée
void print(int x, int y, const char* ch) {
    gotoxy(x,y);
    printf("%s",ch);
    fflush(stdout);
}

// Animation pour les yeux de la vache
void yeux(){
    for (int i =0; i < 4; i++)
    {
        print(5,14,"--");
        usleep(700000);
        print(5,14,"oo");
        usleep(700000);
    }
    gotoxy(0,0);
}

// Animation pour la langue de la vache
void langue(){
    for (int i =0; i <4 ;i++)
    {
        print(7,14," U");
        usleep(700000);
        print(7,14,"  ");
        usleep(700000);
    }
    gotoxy(0,0);

}
```

```
// Animation pour la vache se déplaçant vers l'arrière
void vachearriere(){
    char patte[50];
    int x=1;
    int y=0;

    for(int i=0;i<15;i++){
        int cpt=0;
        y++;
        while (cpt!=3){
            if (cpt==3){
                strcpy(&patte[0],"\\\\");
            }
            else if (cpt==2){
                strcpy(&patte[0],"||");
            }
            else {
                strcpy(&patte[0],"//");
            }
            gotoxy(4,y);
            printf("            ^__^\n");
            gotoxy(5,y);
            printf("            (oo)\\_______\n");
            gotoxy(6,y);
            printf("            (__)\\       )\\/\\\n");
            gotoxy(7,y);
            printf("                ||----w |\n");
            gotoxy(8,y);
            printf("                %s     %s\n",patte,patte);
            cpt++;
            usleep(70000);
            update();

            }
        }
    }

// Animation pour la vache marchant vers l'avant
void vachemarche(){
    char patte[50];
    int x=1;
    int y=15;

    for(int i=0;i<15;i++){
        int cpt=0;
        y--;
        while (cpt!=3){
            if (cpt==0){
                strcpy(&patte[0],"\\\\");
            }
            else if (cpt==1){
                strcpy(&patte[0],"||");
            }
```

```
            else {
                strcpy(&patte[0],"//");
            }
            gotoxy(4,y);
            printf("            ^__^\n");
            gotoxy(5,y);
            printf("            (oo)\\_______\n");
            gotoxy(6,y);
            printf("            (__)\\       )\\/\\\n");
            gotoxy(7,y);
            printf("                ||----w |\n");
            gotoxy(8,y);
            printf("                %s     %s\n",patte,patte);
            cpt++;
            usleep(70000);
            update();

            }
        }
    }

// Fonction pour faire marcher la vache
void marche(){
    vachemarche();

}


// Fonction principale
int main(int argc,char *argv[]) {
    update(); // Efface l'écran
    options(argc,argv); // Gestion des options
    yeux();  // Animation pour les yeux
    fflush(stdout);  // Nettoyage du flux de sortie
    update();  // Efface l'écran
    options(argc,argv); // Gestion des options
    langue(); // Animation pour la langue
    update(); // Efface l'écran
    vachearriere(); // Animation pour la vache se déplaçant vers l'arrière
    marche(); // Animation pour la vache marchant
    return 0;
}
```

reading_cow.c

Après avoir utiliser make pour compiler tous les fichiers dans le bonne ordre, il faut entrer dans l'interpréteur de commande './reading_cow' avec un fichier texte (il y a un fichier texte pour tester le programme et il se nomme 'fichier_test') pour éxécuter ce programme

Ce code à pour objectif est de faire lire un fichier texte à la vache, caractère par caractère: chaque caractère apparait alors successivement dans la gueule de la vache et est avalé et alors placé à la suite des caractères déjà avalés dans la bulle de texte. Il y a une seconde de délai entre chaque caractère affiché.

Exemple d'utilisation : ./reading_cow fichier_test

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

// Fonction pour effacer l'écran
void clear_screen() {
    printf("\033[H\033[J");
}

// Fonction pour déplacer le curseur à la position spécifiée
void gotoxy(int x, int y) {
    printf("\033[%d;%dH", x, y);
}

// Fonction pour afficher un message dans une bulle
void message(char *bubble) {
    int len = strlen(bubble);
    // Affichage de la ligne supérieure de la bulle
    gotoxy(1, 1);
    printf(" ");
    for (int i = 0; i < len + 2; i++) {
        printf("_");
    }
    printf(" \n");
    // Affichage du texte dans la bulle
    printf("< %s >\n", bubble);
    // Affichage de la ligne inférieure de la bulle
    printf(" ");
    for (int i = 0; i < len + 2; i++) {
        printf("-");
    }
    printf("\n");
}

// Fonction pour afficher la vache
void vache() {
    printf("        \\   ^__^\n"
           "         \\  (oo)\\_______\n"
           "            (__)\\       )\\/\\\n"
           "                ||----w |\n"
           "                ||     ||\n");
}

// Fonction pour lire et afficher le contenu d'un fichier avec une vache lisant
void reading_cow(FILE *file) {
    clear_screen(); // Efface l'écran
    char character;
    char bubble[100] = ""; // Initialise une chaîne de caractères pour la bulle de
texte
    message(bubble); // Affiche la bulle vide
    vache(); // Affiche la vache
```

```
    int x_bouche = 7; // Position x de la bouche de la vache
    int y_bouche = 14; // Position y de la bouche de la vache

    // Boucle pour lire et afficher le contenu du fichier
    while ((character = fgetc(file)) != EOF) {
        if (character == '\n') {
            continue; // Ignore les caractères de nouvelle ligne
        }

        gotoxy(x_bouche, y_bouche); // Déplace le curseur à la position de la
bouche de la vache
        printf(" "); // Efface le caractère précédent

        gotoxy(x_bouche, y_bouche); // Déplace le curseur à la position de la
bouche de la vache
        printf("%c", character); // Affiche le caractère
        fflush(stdout);  // Vide le tampon de sortie
        sleep(1);   // Attend une seconde pour simuler la lecture

        int len = strlen(bubble);
        // Ajoute le caractère à la chaîne de caractères de la bulle
        bubble[len] = character;
        bubble[len + 1] = '\0';
        message(bubble); // Met à jour la bulle de texte avec le nouveau contenu
    }

    clear_screen(); // Efface l'écran
    message(bubble); // Affiche la bulle de texte final
    vache(); // Affiche la vache
}

// Fonction principale
int main(int argc, char *argv[]) {
    FILE *file;
    // Vérifie si le nombre d'arguments est correct
    if (argc != 2) {
        printf("Usage: ./a.out <fichier>\n");
        return 1;
    }
    // Ouvre le fichier en lecture
    file = fopen(argv[1], "r");
    if (!file) {
        perror("Problème sur le fichier entré en argument");
        return 1;
    }
    // Appelle la fonction pour lire et afficher le contenu du fichier avec la
vache lisant
    reading_cow(file);

    fclose(file); // Ferme le fichier

    return 0;
}
```

Automates

tamagoshi_cow.c

Après avoir utiliser make pour compiler tous les fichiers dans le bonne ordre, il faut entrer dans l'interpréteur de commande './tamagoshi_cow' pour éxécuter et donc jouer

Il s'agit d'un petit jeu vidéo dans lequel il faut nourrir une vache afin qu'elle survive aussi longtemps que possible(le fait de nourir la vache en fonction du stock de nouriture est le seul impact que nous avons sur la vache, sinon le reste comme le crop et la digestion est une question de hasard). Le score du joueur à la fin du jeu est précis´ement ce temps de survie

Exemple d'utilisation : ./tamagoshi_cow

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "options.h"

// Définition des états de la vache
#define LIFEROCKS 2
#define LIFESUCKS 1
#define BYEBYELIFE 0

// Variables globales pour le stock et la forme physique de la vache
int stock = 5;
int fitness = 5;

// Fonction pour afficher la vache en fonction de son état
void affiche_vache1(int etat) {
    switch(etat) {
        case LIFEROCKS:
            affiche_vache("","oo", "","  ",1);
            break;
        case LIFESUCKS:
            affiche_vache("","--", "","  ",1);
            break;
        case BYEBYELIFE:
            affiche_vache("","xx", "","  ",1);
            break;
    }
}

// Fonction pour mettre à jour l'état de la vache
int maj_etat(int etat) {
    if (etat == LIFEROCKS) {
        if (((fitness >= 1 ) && (fitness <= 3))||((fitness  >= 7 ) && (fitness <=
9)))
            return LIFESUCKS;  // Si la forme physique est mauvaise
        else if ((fitness >= 4 ) && (fitness <= 6))
```

```
            return LIFEROCKS; // Si la forme physique est correcte
        else if (fitness <= 0)
            return BYEBYELIFE;  // Si la vache est morte de faim
        else if (fitness >= 10)
            return BYEBYELIFE; // Si la vache est morte de suralimentation

    } else if (etat == LIFESUCKS) {
        if (fitness >= 10)
            return BYEBYELIFE; // Si la vache est morte de suralimentation

        else if (fitness <= 0)
            return BYEBYELIFE;  // Si la vache est morte de faim

        else if (((fitness >= 1 ) && (fitness <= 3))||((fitness  >= 7 ) &&
(fitness <= 9)))
            return LIFESUCKS;  // Si la forme physique est mauvaise

        else
            return LIFEROCKS; // Si la forme physique est correcte
    }
    return etat;
}

// Fonction pour mettre à jour la forme physique de la vache
void fitness_update(int lunchfood, int digestion) {
    fitness = fitness + lunchfood + digestion;
    if (fitness < 0)
        fitness = 0; // Limite inférieure de la forme physique
    else if (fitness > 10)
        fitness = 10; // Limite supérieure de la forme physique
}

// Fonction pour mettre à jour le stock de nourriture de la vache
void stock_update(int lunchfood, int crop) {
    stock = stock - lunchfood + crop;
    if (stock < 0)
        stock = 0; // Limite inférieure du stock de nourriture
    else if (stock > 10)
        stock = 10; // Limite supérieure du stock de nourriture
}

int main() {
    srand(time(NULL)); // Initialisation pour la génération de nombres aléatoires

    int etat = LIFEROCKS; // État initial de la vache
    int duree_de_vie = 0;   // Initialisation du compteur de durée de vie
    int lunchfood, digestion, crop; // Initialisation des variables lunchfood,
digestion, crop

    while (etat != BYEBYELIFE) { // Tant que la vache est en vie

        affiche_vache1(etat); // Affichage de l'état de la vache
```

```
        printf("Stock : %d\n", stock); // Affichage du stock de nourriture
disponible

        // Demande de la quantité de nourriture à donner
        printf("Quantite de nourriture a donner (0-%d) : \n", stock);
        scanf("%d", &lunchfood);
        while (lunchfood < 0 || lunchfood > stock) {
            printf("Quantite invalide, veuillez entrer une valeur entre 0 et %d :
\n", stock);
            scanf("%d", &lunchfood);
        }

        // Génération de valeurs aléatoires pour la digestion et la récolte
        digestion = -3 + rand() % 4;
        crop = -3 + rand() % 7;

        // Mise à jour de la forme physique et du stock de nourriture de la vache
        fitness_update(lunchfood, digestion);
        stock_update(lunchfood, crop);

        // Mise à jour de l'état de la vache en fonction de sa forme physique
        etat = maj_etat(etat);

        // Incrémentation de la durée de vie de la vache
        duree_de_vie++;
    }
    // Affichage de l'état final de la vache
    affiche_vache1(etat);
    // Affichage du score final (durée de vie)
    printf("Votre score final (duree de vie) est : %d\n", duree_de_vie);

    return 0;
}
```

Markfile

pour tout compiler correctement il suffit de taper 'make' dans l'interpréteur de commande puis :

pour utiliser le programme de newcow -> il faut entrer './newcow' puis mettre les bon argumentsn (les argument à mettre sont expliqué au niveau de newcow)

pour utiliser le programme de wildcow -> il faut entrer './wildcow' puis mettre les bon arguments (les argument à mettre sont expliqué au niveau de newcow)

pour utiliser le programme de reading_cow -> il faut entrer './reading_cow' puis mettre un fichier en argument

pour utiliser le programme de tamagoshi_cow -> il faut entrer './tamagoshi_cow', pas besoin d'argument

pour suppimer tous les fichier en trop a la fin -> il faut utiliser la commande 'make clean'

```
all: newcow wildcow reading_cow tamagoshi_cow

newcow: newcow.o option.o
	clang -o newcow newcow.o option.o

wildcow: option.o wildcow.o
	clang -o wildcow option.o wildcow.o

reading_cow: reading_cow.o
	clang -o reading_cow reading_cow.o

tamagoshi_cow: tamagoshi_cow.o option.o options.h
	clang -o tamagoshi_cow tamagoshi_cow.o option.o

newcow.o: newcow.c options.h
	clang -Wall -Werror -Wno-unused -g -DDEBUG -c newcow.c

option.o: option.c options.h
	clang -Wall -Werror -Wno-unused -g -DDEBUG -c option.c

wildcow.o: wildcow.c options.h wildcow.h
	clang -Wall -Werror -Wno-unused -g -DDEBUG -c wildcow.c

reading_cow.o: reading_cow.c
	clang -Wall -Werror -Wno-unused -g -DDEBUG -c reading_cow.c

tamagoshi_cow.o: tamagoshi_cow.c options.h
	clang -Wall -Werror -Wno-unused -g -DDEBUG -c tamagoshi_cow.c

clean:
	rm *.o newcow wildcow reading_cow tamagoshi_cow
```