TP1 : Ghostbusters
Programmation temps-réel

Groupe 2
Orphée Antoniadis Raed Abdennadher Steven Liatti

20 avril 2017

1 Introduction

Le but de ce travail était de réaliser un jeu qui est un mélange entre deux anciens jeux vidéo : le Pacman et le Casse-brique. L'objectif était d'apprendre à utiliser un RTOS et de comprendre son fonctionnement en mode coopératif. Pour se faire nous avions à disposition la carte Mylab2 accompagnée de la librairie FreeRTOS contenant les primitives du RTOS ainsi que la librairie myLab_lib contenant des fonctions utilitaires pour communiquer avec les périphériques de la carte.

2 Intêret de l'utilisation d'un RTOS

*Expliquez brièvement, selon vous, l'intérêt à utiliser un RTOS dans le cadre de ce projet.

3 Structure du programme

faire un shéma bloc

4 Description des tâches

4.1 Gestion du jeu

```
void game_task(void *arg) {
  while(1) {
    while (!ball->active) {
      lcd_print(65, 160, SMALLFONT, FONT_COLOR, BACKGROUND_COLOR, "Press joystick");
      joystick_handler(check_start, TRIGGER);
      SLEEP(10);
    }
    lcd_print(65, 160, SMALLFONT, BACKGROUND_COLOR, BACKGROUND_COLOR,
"Press joystick");
    xSemaphoreGive(sem_ball);
    xSemaphoreTake(sem_game, portMAX_DELAY);
  }
}
```

Lorsque la partie commence, le joueur dois appuyer sur le joystick pour commencer le jeu et lancer la balle. C'est cette tâche qui s'occupe de la gestion de cette fonctionnalité. Pour détécter la pression du joystick, nous avons été obligé de passer par une attente "semi-passive". La tâche va vérifier toutes les 10ms l'état du joystick puis se mettre en attente passive et laisser la main aux autres tâches. Lorsque la joystick est appuyé, la tâche initialise le nombre de vies ainsi que le score du joueur, sort de sa boucle, débloque le sémaphore sur lequel attendait la tâche de la balle puis attend lui même sur un autre sémaphore qui sera débloqué lorsque le joueur aura perdu la balle 3 fois.

4.2 Gestion de la balle

```
  uint8_t collision_id = test_collision(0,object,1,GHOST_NB);
  uint8_t random = rnd_32() % 2;
  if (collision_id != 0 && object[collision_id].active) {
    direction temp[4] = {NORTH, SOUTH, 0, 0};
    temp[3 - (ball->dir & NORTH)] = ball->dir ^ (ball->dir | WEST | EAST);
    temp[2 + (ball->dir & NORTH)] = ball->dir & (WEST | EAST);
    ball->dir = (temp[random] | temp[random + 2]);
    object[collision_id].active = false;
    score+=10;
    DISPLAY_MENU();
  }
}
```

```
/**
 * @brief     This function must be called when there is a collision between
 *          the bottom of the screen and the ball. It decrements the lives
 *          and checks if the player has more than 0 lives. If he does,
```

4.3 Déplacement des fantômes

```
 *              another ghost and the border screen and hide/display after a random
time.
 */
void ghost_task(void *arg) {
  ghost_t *ghost = (ghost_t*)arg;
  uint8_t change_dir = 0;
  uint8_t change_img = 0;
  uint8_t random;
  uint16_t x, y;
  while(1) {
    while(ghost->obj->active) {
      if (change_dir == 100) {
        ghost->obj->dir = direction_map[rand_direction()];
        change_dir = 0;
      }
      if (change_img == 5) {
        animate(ghost);
        change_img = 0;
      }
      x = ghost->obj->x;
      y = ghost->obj->y;
      ghost_ghost_collision(ghost->id);
      display_ghost(ghost);
      if (ghost_left_collision(ghost->obj)) ghost->obj->dir ^= (WEST | EAST);
      if (ghost_right_collision(ghost->obj)) ghost->obj->dir ^= (WEST | EAST);
      if (ghost_up_collision(ghost->obj)) ghost->obj->dir ^= (NORTH | SOUTH);
      if (ghost_down_collision(ghost->obj)) ghost->obj->dir ^= (NORTH | SOUTH);
      move_object(ghost->obj);
      SLEEP(ghost->speed);
      if (ghost->obj->active) update_ghost(ghost, x, y);
      else clear_ghost(x, y);
      change_dir++;
      change_img++;
    }
    SLEEP(20);
    random = rnd_32() % 100;
    if (random < 1) ghost->obj->active = true;
```

4.4 Gestion de la raquette

```
void racket_task(void *arg) {
  uint16_t last_x = racket.x;
  uint16_t last_y = racket.y;
  lcd_filled_rectangle(racket.x, racket.y, racket.x + racket.width, racket.y +
racket.height, RACKET_COLOR);
  while(1) {
    if (joystick_left_pressed(last_x, last_y) || joystick_right_pressed(last_x,
last_y)) {
      last_x = racket.x;
      last_y = racket.y;
```

```
      SLEEP(8);
    } else {
      SLEEP(10);
    }
  }
}
```

5 Analyse des traces

6 Limite du nombre de fantômes

Ralentissement du jeu avec un nombre de traces élevée. La fonction de récupération des traces n'est pas assez performante et le buffer prend un certain temps pour se vider.
Le jeu reste bloqué lors de plus de 21 tâches en parallèle. Nous pensons que cette limite vient de la mémoire alouée à FreeRTOS. AU delà de 21 tâches, on dépasse cette mémoire définie par la macro configTOTAL_HEAP_SIZE définie ici à 20 * 1024. En augmentant cette variable nous pouvons augmenter le nombre de fantomes jusqu'à la limite de la mémoire de la carte. La limite que nous avons atteint est de configTOTAL_HEAP_SIZE = 27 * 1024 avec un nombre de tâches de 29 (sans compter VApplicationIdleHook). Au dessus nous avons un hardfault que nous pensons être due à un dépassement de mémoire de la carte.