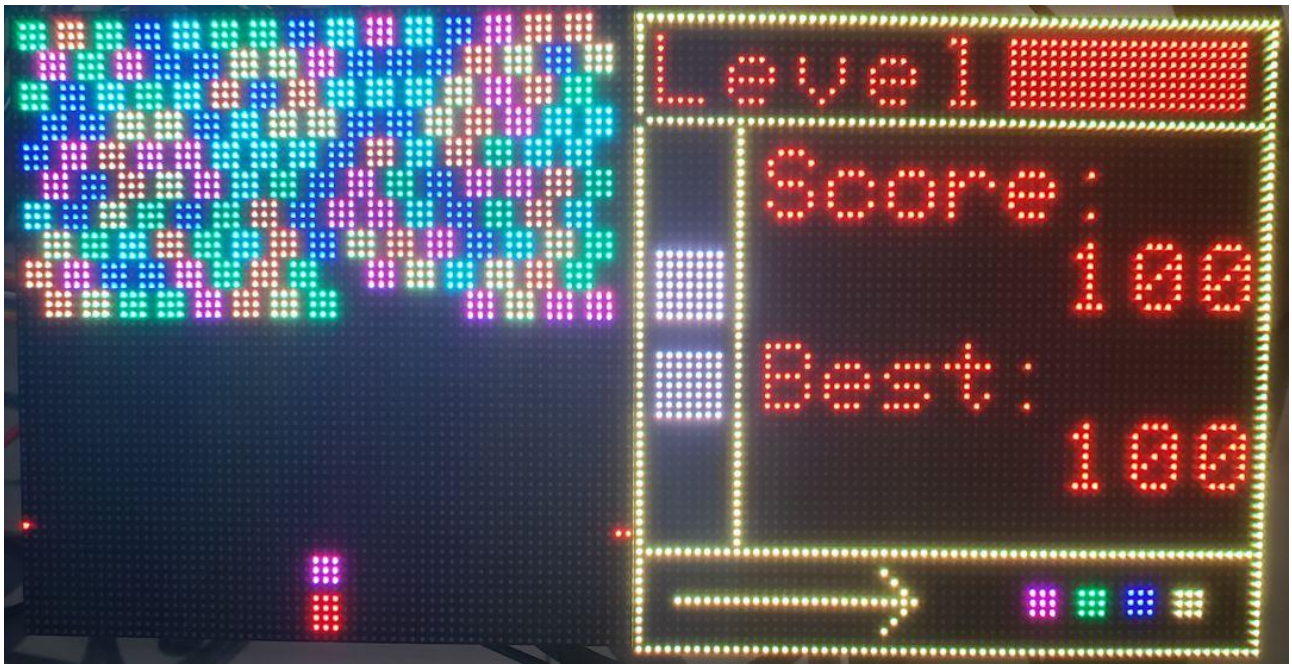
Par Maxime Rieux et Renan Cabane

Compte rendu
jeu de Bubble Shooter
Arduino



I) Introduction

Nous avons codé un jeu de Bubble Shooter sur deux matrices 64x64, l'une pour afficher l'interface et l'autre pour afficher le jeu. Chaque matrice est gérée par une carte Arduino. Celles-ci communiquent entre elles pour échanger le score, la prochaine couleur, le nombre de tentatives restantes au joueur avant que la grille de bille ne descende et les commandes utilisateurs.

Une première interface permet d'informer l'utilisateur qu'il faut sélectionner la difficulté avec « q » et « d » puis appuyer sur espace pour jouer.

II) Partie jeu

Pour le jeu, on peut incliner le canon vers la droite ou vers la gauche avec les touches « e » et « a » respectivement, le déplacement vers la droite avec « d », le déplacement vers la gauche avec « q » et tirer avec « z ». On utilise un terminal externe pour envoyer les commandes vers l'Arduino « jeu » avant de les traiter. Lors d'un tir, la bille se déplace alors jusqu'à collision avec un mur ou une autre bille puis se replace correctement dans la grille. Deux algorithmes sont alors lancés successivement : le premier permet, à l'aide d'un parcours en largeur depuis la bille lancée, de calculer quels sont ses voisins de la même couleur et combien elle a de voisins. Si un paquet de plus de 3 billes est alors formé, on les fait clignoter pour informer l'utilisateur qu'elles vont disparaître, puis on les supprime. Le deuxième permet de vérifier, à l'aide d'un problème de graphe, s'il y a une ou plusieurs billes qui sont isolées (non reliées au sommet du jeu). On garde allumées seulement les billes accessibles depuis le sommet du jeu, les autres clignotent avant de disparaître.

On peut choisir le nombre de couleurs avec lesquelles on veut jouer, le nombre d'essais avant que le jeu ne descende et de combien de lignes le jeu va descendre.
Si une bille vient à se bloquer en dessous des pixels rouges sur les côtés, elle clignote pour indiquer à l'utilisateur que la partie est perdue et toutes les commandes sont ignorées sauf la touche « espace » pour revenir à la sélection de la difficulté de jeu. Un nouvel « espace » relancera une nouvelle partie.

Difficultés rencontrées :

- Un gros manque de mémoire qui nous a obligés à regrouper des tableaux, mettre en commun des variables et des tableaux, optimiser l'utilisation de la mémoire.

- Les structures de données restreintes : il n'y a pas de file, de liste et d'ensemble. Nous avons dû transformer deux tableaux en files avec des indices de début et de fin de file.
- Communication lente entre les cartes Arduino qui nous à obligé à faire attendre l'utilisateur que les échanges de données soient terminés afin que le jeu fonctionne correctement.
- Gérer correctement les conditions de replacement et de collision.
- L'impossibilité de mettre un « delay » dans un timer, ce qui nous a obligé à utiliser un second timer.

Fonctions :

- transmettre_score : permet de calculer le score en fonction du nombre de billes éclatées puis le transmet à la deuxième matrice.
- clignoter : fonction utilisée par un timer, elle permet de faire clignoter les billes lors de la fin de la partie, d'une explosion ou d'une détection de billes isolées.
- set : nous avons dû regrouper deux matrices, l'une de int et l'autre de booléen en une seule. Le « %32 » nous donnait le int et le test « ≥ 32 » le booléen. Pour ce faire, la méthode « set » permet d'ajouter ou de retirer 32 à la case voulue, c'est-à-dire de passer le booléen à vrai ou faux.
- get : permet sur le même principe que le set de retourner la valeur du booléen dans la case voulue.
- boules_isolees : permet, via un algorithme de calcul des sommets accessibles depuis le sommet du jeu, de supprimer les billes qui ne sont pas rattachées au haut du jeu.
- descendre : permet de descendre toutes les billes du jeu de la taille voulue en ajoutant de nouvelles billes.
- perdu : méthode appelée lorsque le joueur a perdu la partie.
- init_interface : permet d'initialiser l'interface.
- afficher_jeu : permet d'afficher le jeu depuis la matrice de jeu après une descente par exemple, ou encore au début du jeu.
- debut_bubble : permet de guider l'utilisateur en lui indiquant sur quelle touche appuyer pour lancer la partie.
- initialisation_jeu : permet de lancer une nouvelle partie, créer une grille aléatoire, réinitialiser les paramètres et effacer la partie précédente.
- estPossible : renvoie un booléen indiquant si le déplacement est possible afin de rester dans la taille de la matrice.
- case_libre : renvoie un booléen qui indique si le prochain emplacement du cube est libre ou occupé (s'il y a collision avec un autre cube).
- exploser : permet via un parcours en largeur, de calculer le paquet de tous les voisins de l'emplacement d'arrivée de la bille ayant la même couleur que celle-ci.
- deplacer_cube : permet de déplacer le cube dans la direction souhaité. Cette fonction est appelée par un timer.

- deplacer : permet de déplacer le canon et de l'incliner dans la direction souhaitée.
- rearmer : permet de réafficher une bille devant le canon une fois que la précédente est traitée.

III) Partie interface utilisateur

La carte Arduino de la matrice interface reçoit les commandes de celle du jeu. Si la partie n'est pas en cours, l'appui sur les touches modifiera la difficulté du jeu en la diminuant (touches « q ») ou en l'augmentant (touches « d »).

L'affichage « Best » sur la matrice indique le meilleur score du joueur pour la session de jeu en cours. L'affichage de « Score » indique le score du joueur pour la partie en cours, ou pour la partie précédente s'il y en a eu une dans la session de jeu en cours.

Le rectangle centré à gauche de la matrice indique le nombre de coups restants au joueur pendant la partie avant que les billes ne descendent d'une case.

Le rectangle en bas de la matrice indique la file des billes à suivre pendant la partie en cours.

Difficultés rencontrées :

Nous voulions initialement adapter une communication radio entre les deux cartes Arduino, mais les difficultés à faire fonctionner ce type de communication et l'intérêt limité de cette solution dans notre contexte nous ont finalement décidés à opter pour la solution de la communication en série.

Nous voulions afficher le score et ses mises à jour en temps réel de façon à ce que le score soit incrémenté de 1 points par 1 points avec un timer, mais de nombreux bugs visuels faisant varier l'intensité de la luminosité des LEDs nous ont poussés à n'incrémenter les points que de 10 en 10 afin de limiter l'intervalle de rafraichissement du timer. Ceci dans l'objectif que l'actualisation du score reste visuellement sympathique à voir et qu'elle se fasse dans un temps raisonnable.

Fonctions :

- initAnim : au démarrage du programme, lance l'animation de démarrage du jeu.
- initInterface : après initAnim, et à la fin d'une partie, affiche l'interface permettant le choix de la difficulté.
- reinitialisationScore : remet la variable score à 0 et refait l'affichage de celui-ci dans la matrice.

- majScore : met à jour le score en temps réel pendant la partie. Appelé par le Timer3.
- choixDifficulte : modifie la difficulté de la partie et la jauge de difficulté dans la matrice interface.
- nbEssai : met à jour les cubes indiquant le nombre d'essais restants pour le joueur.
- dernierEssai : si le nombre de coups restants au joueur avant que la grille de billes descende est égale à 1, fait clignoter le cube le plus bas du rectangle de gauche dans la matrice interface. Fait également clignoter la prochaine bille que recevra le joueur une fois son coup en cours joué.
- creerFile : crée une nouvelle file de couleurs pour les billes au lancement d'une nouvelle partie et l'affiche dans le rectangle du bas dans la matrice interface.
- pop : renvoie le numéro de la couleur de la prochaine bille que recevra le joueur une fois son coup en cours joué.