Université Gustave Eiffel

L2 – Programmation C

Three To Go

Projet du S3

Clément GAUDET / Maxime JAILLARD
16/01/2021

Sommaire

1 MANUEL UTILISATEUR

1.1 COMPILATION

Pour compiler le programme, une fois le projet décompressé, il suffit de se mettre à la racine et de taper la commande « **make** ».

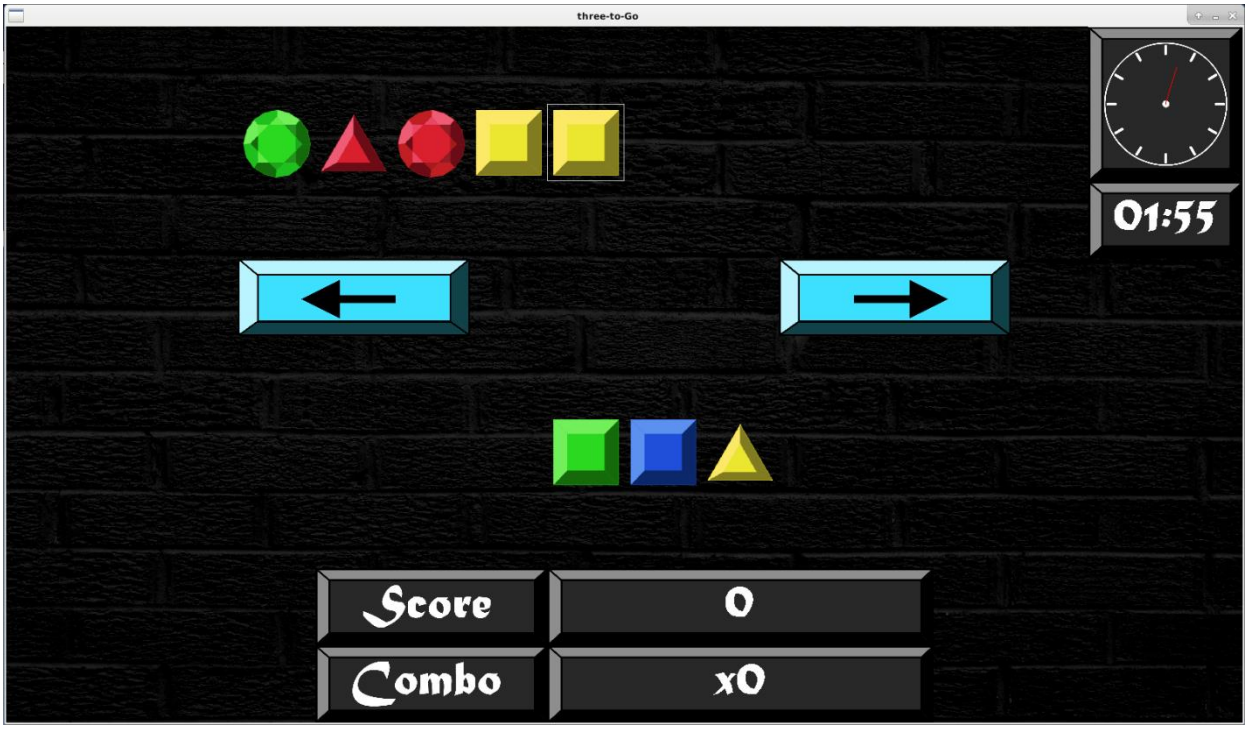
La librairie MLV est nécessaire pour la compilation, et celle-ci doit être faite sous Linux.

Une fois la compilation terminée, un exécutable nomme « **threetogo** » est créé, qui peut être lancé.

1.2 UTILISATION

Three To Go est **un jeu de puzzle.** Le but est de **mettre côte à côte des gemmes de la même couleur et/ou de la même forme** afin de les faire disparaître et de gagner des points, le tout sur un temps imparti **de 2 minutes**.

En lançant l'exécutable, l'écran-titre s'affiche en premier. Celui-ci montre notamment la liste des 5 meilleurs scores réalisés par le passé. Un clic de la souris permet de lancer le jeu.



1. Capture de l'interface du jeu

Voici l'interface du jeu. Celle-ci est composées de plusieurs parties. Au centre se trouve **l'aire de jeu**. C'est là où les gemmes qu'il faut combiner. Au-dessus, on trouve la liste des 5 prochaines gemmes à mettre en jeu, la prochaine étant entourée d'un carré blanc. En cliquant sur les **boutons fléchés** de part et d'autre, on pourra **insérer cette prochaine gemme à gauche ou à droite de l'aire de jeu.**

Il est important de savoir **qu'un maximum de 16 gemmes** peuvent tenir dans l'aire principale. Une fois ce nombre atteint, il est impossible d'ajouter plus de gemmes et les boutons d'insertions disparaîtront. La deuxième option de jeu est de **cliquer sur une des gemmes déjà en jeu**. Deux nouvelles gemmes apparaitront alors. Cliquer sur celle du haut effectuera un **décalage circulaire vers un gauche de toutes les gemmes de la même couleu**r. Celle du bas fera **le même décalage pour les gemmes de la même forme.** Cette option est toujours disponible, tant qu'au moins une gemme est en jeu.

L'interface affiche également une **horloge** dans le coin supérieur droit, indiquant le temps restant, et **un indicateur du score et du combo actuel** en bas de l'écran.

Le système de score fonctionne comme suit :

Chaque gemme retirée comme faisant partie d'une séquence de gemmes de la même couleur ou de la même forme vaut **100 points de base**. Les séquences de même couleur et de la même forme sont comptées séparément. Ainsi, si 3 gemmes de la même couleur ET de la même forme sont retirées, ce coup vaudra (100 * 3) * 2 = 600 points.

Quelques autres règles s'appliquent :

- Les combinaisons réalisées en effectuant un **décalage circulaire valent double en points.**
- Si les gemmes supprimées par la réalisation d'une **combinaison permettent de former une nouvelle combinaison au cours du même coup, les points de celle-ci sont doublés**. Si d'autres combinaisons sont encore formées, cette logique continue à s'appliquer. Le nombre de points des combinaisons suivantes sera alors multiplié par 4x, 8x, 16x, etc.
- **Chaque coup d'affilé éliminant au moins une combinaison** permet d'augmenter le compteur de combo. Celui-ci est **un autre multiplicateur s'appliquant au score du coup effectué**.

Tous ces multiplicateurs se multiplient ensemble. Il est donc possible de réaliser des coups ramenant beaucoup de points en les associant.

Des **effets sonores** joueront dans certains cas, notamment :

- Première combinaison de la partie
- Combos
- Coups valant plus de 1000 points

Une fois les 2 minutes écoulés, **un écran de fin de partie s'affiche avec le score final**. De plus, un message supplémentaire s'affichera si un nouveau meilleur score a été réalisé. **Celui-ci cela alors automatiquement sauvegardé. Un clic final permettra de fermer le jeu.**

2 MANUEL TECHNIQUE

2.1 ORGANISATION GÉNÉRALE DU PROGRAMME

Le programme est rangé dans le dossier src, et est subdivisé en un certain nombre de fichiers :

- **main.c** : contient la fonction main. Elle s'occupe d'exécuter les diverses fonctions permettant de charger et initialiser les ressources nécessaires au jeu, puis d'exécuter celles permettant de faire tourner le jeu, et enfin celles libérant la mémoire
- **threetogo.c :** contient les fonctions principales du programme, notamment la boucle principale
- **token.c** : contient les fonctions permettant de manipuler directement la structure de liste chaînée utilisée par le programme
- **moteur.c** : contient les fonctions composant le moteur du jeu, bon nombre de fonctions clés s'y trouvent
- **graphique.c :** contient les fonctions gérant à la fois l'affichage, et l'interaction du joueur avec l'écran
- **audio.c :** contient les quelques fonctions gérant le son
- **fileio.c** : contient les quelques fonctions permettant de lire et enregistrer les high score dans un fichier

Chaque fichier, sauf main.c, a bien sûr son propre header associé, et chacun n'inclus que les headers dont il a besoin individuellement pour être compilé.

Une documentation générée par Doxygen est incluse. Elle se trouve dans /doc/html/index.html

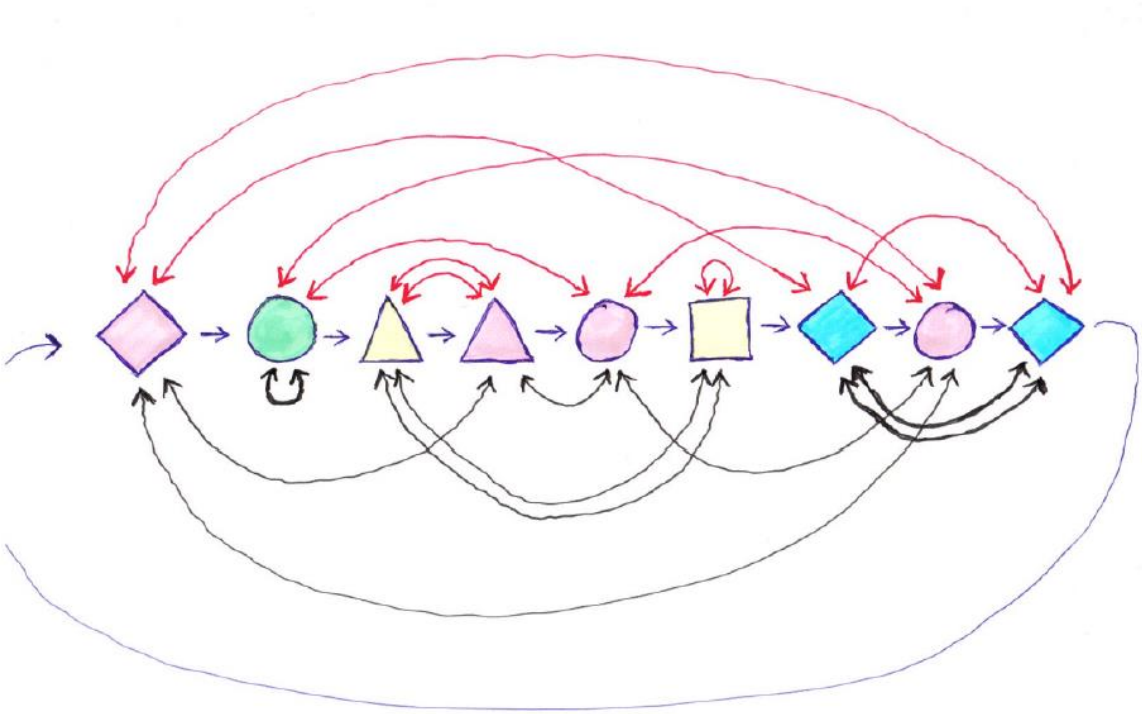
2.2 STRUCTURES DE DONNÉES

Le programme s'appuie sur une donnée de structure principale, celle permettant de représenter des listes de gemmes (appelées tokens dans le code).

Il s'agit d'une **liste chainée relativement complexe.** Le chaînage principal est **simple circulaire**. Il suit l'ordre des tokens. A noter, que **le type Liste pointe vers le dernier élément du chainage simple**. En effet, cela permet un **accès rapide au premier et au dernier élément de la liste** à l'aide d'une seule adresse.

En plus de ce simple chaînage, s'ajoute deux **doubles chaînages circulaires**. Ceux deux chainages doublent relient les tokens d'une même couleur, et les tokens d'une même forme. Ces doubles chaînages sont utilisés pour **faciliter certaines opérations, notamment le décalage circulaire des tokens**.

La liste principale de tokens utilise cette structure, mais également la liste des 5 tokens suivants à placer, bien que celle-ci n'utilise pas le double chainage, qui ne sert à rien dans ce cas.



2. Schéma représentant le chainage des tokens

L'avantage est que le token est créé une fois, quand il apparait dans la queue, puis le même est utilisé une fois mis dans la liste principale jusqu'à sa suppression. Il suffit de changer les chainages.

La gestion des doubles chaînages étant compliquée, elle est souvent faite **dans un second temps**, après la manipulation du simple chainage. Dans des cas simples tels que l'extraction d'un élément de la liste, le double chaînage n'est mis à jour que localement, **mais dans les cas plus complexes tels que le décalage circulaire, la totalité du double chaînage affecté est mis à jour.**

D'un point de vue mémoire, les éléments de la liste chaînée sont **alloués dynamiquement sur le tas**, et le restent tant que le token apparaît dans l'aire de jeu. **Une fois le token éliminé de la liste dans une combinaison, la mémoire est libérée**. Tous les tokens restants sont également libérés à la fermeture du programme, permettant d'éviter toute fuite à ce niveau.

La structure de données Token est définie dans threetogo.h, et les fonctions manipulant la liste directement sont trouvées dans token.h.

2.3 MOTEUR DU JEU

moteur.c contient les fonctions composant le moteur du jeu.

Certaines sont relativement simples, telles que celles permettant d'initialiser une liste, d'en libérer une, ou alors d'ajouter un token à droite ou à gauche, et il n'est pas nécessaire de revenir dessus.

Trois autres fonctions sont cependant **bien plus importantes et complexes** :

- **check_combinations** : cette fonction est en charge de vérifier les combinaisons présentes dans la liste de tokens, de compter les points obtenus, de supprimer ces tokens de la liste, et de libérer la mémoire associée
- **shift_commonshape_left** et **shift_commoncolor_left** : ces fonctions, qui fonctionnent de la même manière, sont celles qui sont en charge d'effectuer le décalage circulaire de tokens d'une même forme/couleur

2.3.1 int check_combinations(Liste *lst, int mul)

Une structure est définie dans moteur.h juste pour le fonctionnement de cette fonction.
La structure **Sequence** permet de caractériser une **séquence de n tokens d'affilé d'une même caractéristique**. Deux sont utilisées lors du parcours de la liste de tokens, une pour les couleurs, une pour les formes.

La fonction parcourt la liste des tokens en mettant à jour à chaque fois la longueur de la séquence actuelle. Dès lors qu'une séquence **de plus de 3 tokens** se termine, les points sont ajoutés, et **les tokens sont marqués pour suppression dans un tableau de tokens to_delete,** à l'aide d'une fonction auxiliaire **add_addresses**, qui va retrouver l'adresse de chaque token concerné.

Une fois le parcours terminé et la taille des dernières séquences vérifiées**, la fonction extrait ensuite tous les tokens marqués de la liste, et libère la mémoire associée.**

Enfin, si un score supérieur à 0 a été trouvé**, un appel récursif à lui-même est effectué** afin d'éliminer d'éventuelles combinaisons s'étant formées à la suite de la suppression des tokens.

La fonction renvoie alors le score obtenu.

Ce fonctionnement permet **de prendre en compte toute longueur de séquence, y compris des quadruplets et quintuplets,** et prend en compte les autres combinaisons pouvant apparaître après disparition de gemmes.

2.3.2 void shift_commonshape_left(Liste *lst, Token *tok)

Puisque la function jumelle pour les couleurs fonctionne de la même manière, nous ne décrirons ici que celle pour les formes.

Cette fonction effectue le décalage circulaire vers la gauche de tous les tokens de la même forme que tok. Pour cela, elle effectue une **série de transpositions des tokens suivant le double chaînage forme de tok**.

En effet, prenons par exemple la liste **1**, 2, **3**, **4**, 5, 6, **7**. Faire un décalage circulaire des tokens rouges ici revient à faire une permutation qui n'est ni plus ni moins qu'un cycle (1, 3, 4, 7), qu'on peut décomposer en transpositions : (1, 3).(1, 4).(1, 7).

La fonction utilise ce principe et parcourt le double chainage forme, et **effectue à chaque étape une transposition**, ce qui permet à la fin, **d'obtenir le décalage voulu**. L'échange de deux tokens est obtenu par la fonction **swap**, codé dans token.c, qui échange la position de deux tokens. Cette fonction **ne met pas à jour les doubles chaînages cependant.**

Il est inutile de mettre à jour le double chainage forme, puisque celui-ci est inchangé par le décalage. **Cependant**, **les chaînages couleurs, eux, sont mis à jour à la suite des transpositions**. Chaque chaînage est mis à jour complètement pour éviter des cassures de chaînage dans des cas particuliers.

2.4 Généralités sur l'affichage

La librairie MLV est utilisée pour gérer l'affichage du jeu.

D'un point de vue conceptionnel, afin de permettre de positionner et d'aligner facilement les éléments sur la fenêtre, et de pouvoir modifier la taille de la fenêtre si l'on voulait, **il a été décidé de diviser la fenêtre en une grille de 16x9 carrés**. Le côté de chaque carré est déterminé par la longueur de la fenêtre que l'on veut. Le tout est défini à l'aide de constantes que l'on peut retrouver dans threetogo.h. Notamment, la longueur de la fenêtre est **SIZEX** et le côté d'un carré de la grille est **RESO**.

Toutes les fonctions d'affichages utilisent cette grille et donc la constante RESO pour placer et dimensionner les éléments de l'interface.

Le jeu utilise un certain nombre d'images. Celles-ci sont stockées dans le dossier assets, et **sont toutes chargées dans un tableau au début de l'exécution du programme**, permettant leur utilisation.

2.5 Boucle principale

La boucle principale du jeu se trouve dans la fonction **main_loop**, dans threetogo.c.

Une structure **Game** est utilisée ici afin de plus facilement transférer un certain nombre de paramètres important de la partie entre fonctions. Celle-ci contient notamment le **score actuel, le temps écoulé, le compteur de combo, ainsi que la liste des tokens en jeu, et celle des tokens suivants**. Elle est initialisée dans le main grâce à une fonction, **game_init**, et la mémoire utilisée et libérer par un appel à la fonction **game_free**.

2.5.1 Gestion des événements

À chaque tour de la boucle principale, **la liste des évènements enregistrés depuis le tour précédent est traitée entièrement**. Seuls les clics du bouton de la souris sont pris en compte, le jeu se jouant uniquement avec la souris.

Selon les coordonnées du clic, le programme détermine **quels boutons ont été cliqués** et agit en conséquence en appelant les fonctions appropriées du moteur (ajout d'un token à gauche ou à droite, décalages circulaires, etc.). En cas de clic sur une gemme, **un second clic sera attendu pour déterminer l'action à prendre** (décalage des formes ou des couleurs).

Si un coup est effectué, le programme vérifiera alors les combinaisons effectuées et comptera les points. Il mettra également à jour le compteur de combo, et jouera des effets sonores selon le contexte.

2.5.2 Affichage

On ne rafraîchit l'affichage qu'une seule fois par occurrence de la boucle. Toutes les fonctionnalités d'affichage sont regroupées dans la fonction **refresh_screen** du fichier graphique.c. Cette fonction commence par effacer tous les dessins de la fenêtre et applique la couleur noire à la totalité de la fenêtre. Elle redessine ensuite dans l'ordre :

- La queue et son cadre de sélection
- Les boutons d'ajout
- La liste des enchaînements de jetons
- Le timer et le score du joueur

Seulement une fois tous ces dessins effectués, on fait appel à la fonction **MLV_actualise_window**. En pratique, cela évite que l'interface et les dessins ne « flashent » lorsque l'on tente de faire fonctionner le jeu à un taux d'affichage par seconde agréable (de l'ordre de 50-60 images par secondes).

La fonction **refresh_screen** tient également compte de l'état du jeu en utilisant les coordonnées du dernier clic enregistré et le nombre de jetons dans la ligne du dessous. Si le clic a eu lieu sur l'un des jetons de la liste, celui-ci sera mis en évidence lors du rafraîchissement par l'ajout des jetons de démonstration au-dessus et en dessous pour représenter

les décalages. Si le nombre de jetons dans la liste atteint le nombre maximum déterminé par la constante **MAX_TOKENS**, les boutons d'ajout ne seront pas dessinés.

La fréquence de rafraichissement est fixée par une constante FRAME_RATE, fixée à 30 par défaut. L'appel à **MLV_delay_according_to_frame_rate** permet d'assurer une fréquence stable.

2.6 GESTION DE LA MÉMOIRE

L'allocation dynamique est utilisée à plusieurs endroits dans le programme, notamment pour les tokens. Toute la mémoire allouée pour ceux-ci **est libérée par des free quand le token est éliminé, que ce soit pendant la partie, ou à la fin.**

De plus, **la mémoire allouée pour la librairie MLV est également libérée**, ce qui inclue, les images, les sons, la police d'écriture, la fenêtre.

Il ne persiste donc **aucune fuite de mémoire causé par notre code en lui-même**. Cependant, des fuites de mémoire persistent lors de la vérification à l'aide de **valgrind**, causées par le fonctionnement de la librairie MLV elle-même. Puisque même simplement ouvrir et libérer une fenêtre entraîne ces fuites-là, il a été décidé qu'il s'agissait d'un problème de la librairie et pas d'un problème lié au programme.

2.7 BUGS

Tous les bugs connus au cours du développement ont été réglés. Au moment du rendu, aucun bug n'a été relevé.

3 RÉPARTITION DU TRAVAIL

Le développement s'est fait **en parallèle** entre les deux membres du binôme, l'un plutôt porté sur le moteur, l'autre sur le développement graphique. Une fois la structure de liste mise en place, le travail s'est fait de façon **concurrente**, ainsi que **collaborative**, avec **utilisation de Git et d'un repo à distance** pour synchroniser le travail fait. L'effort de documentation, et le nettoyage du code à la fin a été un effort conjoint, ainsi que l'écriture du rapport.

Maxime s'est concentré sur l'interface graphique du jeu (dessin des jetons, des boutons, du timer, de l'horloge et affichage du score). On réutilise des fonctions logiques utilisées dans la version à affichage ascii du jeu (utilisée seulement au cours du développement) pour contrôler l'état des différents paramètres de la partie.

La première étape du travail a été de créer plusieurs fonctions et structures élémentaires. Découper la fenêtre en cases semblait plus évident pour travailler de manière plus lisible sur l'affichage des dessins. Il a donc commencé par créer une structure **Case** et la fonction **mouse_to_square** pour transformer tous les clics dans la fenêtre en clics de zone. Une fois que la gestion des coordonnées était terminée, la partie suivante était centrée sur l'interprétation de ces coordonnées et répercuter les actions de l'utilisateur sur la queue et la liste de jetons ajoutés.

Il a fallu à Maxime beaucoup de temps et l'aide de Clément pour la partie gestion des événements. Trouver le moyen de gérer sans la moindre latence chaque occurrence de la boucle après chaque clic de l'utilisateur a été moins évident que je le pensais. Il a fallu explorer la librairie MLV pour pouvoir gérer le framerate et comprendre le format et l'utilité des différents arguments de ces fonctions.

Clément a surtout travaillé sur le moteur du jeu et la manipulation de la structure de liste chainée. Un certain de nombre de fonctions étaient relativement simples à coder, étant des implémentations déjà vues en cours ou connues de façon générale. La gestion des doubles chaînages a été une partie un peu plus délicate, mais plus dans l'aspect conceptionnel, que mise en pratique.

Les fonctions permettant de vérifier les combinaisons et les décalages circulaires ont été les parties les plus dures à coder, demandant de mettre au point des algorithmes pour gérer tous les cas, ce qui a demandé un peu de temps. De plus, des bugs sont apparus plus tard, lors de tests, ce qui a dû entraîner quelques sessions de débogage et quelques réécritures pour régler le problème.

Accessoirement, il s'est également occupé du design graphique et audio du programme.