

RAPPORT DE PROJET : LE JEU DU TAQUIN (SAE11_2025)

Implémentation du Jeu du Taquin avec Découpage d'Image

Auteurs : Baptiste LECOINTRE et Youssef AMROUCHI

07 décembre 2025

Table des Matières

RAPPORT DE PROJET : LE JEU DU TAQUIN (SAE11_2025).....	1
Table des Matières	1
1. Introduction	2
2. Fonctionnalités du Programme.....	3
3. Architecture du Programme et Justification	5
4. Explication des Données.....	7
4.1 Structure de game_state_t et son Contenu :	8
4.2 Conventions de Représentation.....	8
5. Algorithme de Mélange et Preuve de Solvabilité	8
5.1 Méthode Utilisée	9
5.2 Démonstration de Solvabilité	9
6. Conclusion Personnelle	9

1. Introduction

Le Taquin est un jeu de puzzle classique consistant à réordonner des tuiles numérotées dans une grille comportant un seul espace vide dans la version initiale du jeu. Ici le but de notre projet était de développer une version nouvelle version de ce jeu, en remplaçant les nombres par une image découpée en tuiles.

Le programme a été entièrement développé en C89 en utilisant la bibliothèque graphique de l'IUT. Les contraintes majeures portaient sur la modularité du code, le support des interactions à la fois à la souris et au clavier, et surtout, la garantie que toute configuration de jeu mélangée soit solvable. Le travail a été effectué en utilisant un dépôt Git.

2. Fonctionnalités du Programme

Le programme nous offre deux vues principales :

Tout d'abord il y a le menu :

Le joueur est accueilli par une interface où il peut :

- Choisir une image parmi trois aperçus. On a choisi une image de Spiderman, d'un capybara et tout simplement de notre iut. Cela laisse la possibilité de choisir ce qu'on veut avec des thèmes et des couleurs différentes.
- Choisir les dimensions de la grille de façon indépendante, de 3 à 8 lignes et colonnes, à l'aide de boutons cliquables et des touches de direction du clavier. Cela permet donc de choisir la difficulté en quelques sortes.
- Démarrer la partie via le bouton "DÉMARRER".

Et ensuite il y a la partie en jeu :

- Affichage : Le plateau de tuiles découpées et mélangées est affiché. L'image a été découpé 300 fois aléatoirement. L'emplacement vide est marqué en jaune et se situe obligatoirement en haut à gauche à chaque fois.
- Modèle : Le joueur dispose d'une petite image modèle en haut à droite de l'écran pour référence, ne chevauchant pas la zone de jeu. C'est l'image de la photo initiale qu'il a choisi sur le menu.
- Compteur de Coups : Le nombre de coups joués est constamment indiqué en bas à gauche de l'écran.
- Interactions : Les déplacements sont réalisables soit avec la souris sur une tuile à côté, ou alors avec les quatre flèches directionnelles du clavier.
- Fin de Partie : La victoire affiche un message "BRAVO !" et propose au joueur de commencer une Nouvelle Partie (touche 'N') ou de Quitter (touche 'Q'). Le bouton "MENU (R)" permet de revenir à la configuration à tout moment.

Voici par exemple notre menu :

CONFIGURATION DU JEU TAQUIN

1. Choisir l'image :



2. Choisir les dimensions (3x3 à 8x8) :

Lignes : 3

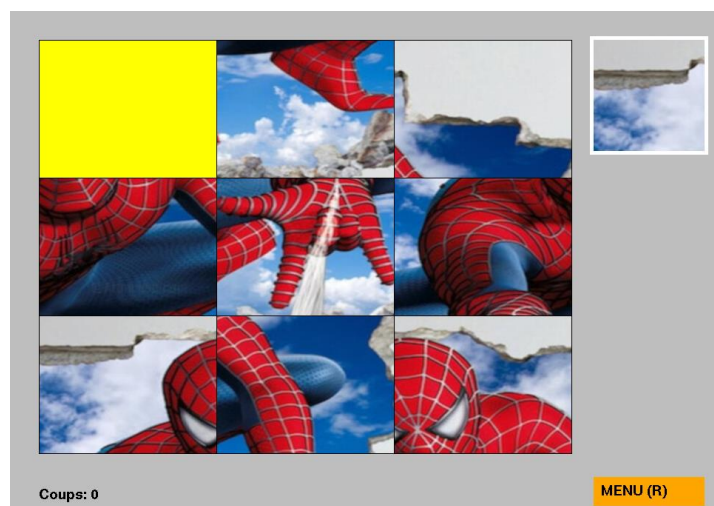


Colonnes : 3



DEMARRER

Et voici par exemple la partie avec l'image de Spiderman :



3. Architecture du Programme et Justification

Le code est divisé en plusieurs fichiers source pour garantir la modularité et la séparation des préoccupations. Permet aussi de mieux se repérer. Voici notre arborescence pour le dossier src ou se situe tout notre code :

Fichier	Rôle (Responsabilité)	Type de Contenu
main.c	Orchestration : Boucle principale (do...while), gestion des transitions (Menu $\rightarrow$ Jeu), initialisation de l'environnement (srand, InitialiserGraphique).	Implémentation
menu_view.c	Vue Configuration : Rendu du menu (images d'aperçu, boutons +/-), gestion des clics souris et des touches clavier dans la phase de configuration.	Implémentation
menu_view.h	Définitions du Menu : Structure menu_state_t et prototypes des fonctions du menu.	Déclaration
game_logic.c	Moteur du Jeu (Logique Pure) : Implémentation de l'état du Taquin, game_init, game_move_tile, game_shuffle_safe, game_is_solved.	Implémentation
game_logic.h	Définitions du Moteur : Structure game_state_t et prototypes de la logique (déplacements, mélange).	Déclaration
game_view.c	Vue Jeu / Boucle d'Interaction : Rendu du plateau, affichage du modèle d'image et du compteur de coups, gestion des 4 flèches directionnelles et du bouton MENU (R).	Implémentation
game_view.h	Définitions de la Vue Jeu : Prototypes de game_render et game_loop.	Déclaration
images.c	Découpage & Ressources : Définition des chemins	Implémentation

	d'accès aux images, calcul des coordonnées pour le découpage (images_slice), gestion des dimensions.	
images.h	Définitions d'Image : Structures tile_t et image_source_t.	Déclaration
utils.c	Utilitaires Math/Aléatoire : Implémentation de abs_val et random_int.	Implémentation
utils.h	Définitions Utilitaires : Prototypes des fonctions mathématiques.	Déclaration
defs.h	Constantes Globales : Définit WINDOW_WIDTH, WINDOW_HEIGHT, REFRESH_CYCLE, et les autres constantes partagées.	Déclaration

4. Explication des Données

L'état complet de notre jeu du Taquin est géré par une seule et unique structure centrale, nommée **game_state_t**. Cette approche est fondamentale pour respecter les bonnes pratiques C89 et les consignes APL

Toutes les informations nécessaires au moteur de jeu, à la vérification de victoire et à l'affichage sont encapsulées dans cette structure.

4.1 Structure de game_state_t et son Contenu :

Voici les principales variables qui sont dans notre jeu :

- **rows (int) et cols (int)** : Ces deux entiers stockent les dimensions actuelles de la grille. Ils sont choisis par le joueur au menu.
- **Permutation (int)** : C'est le cœur du plateau de jeu. C'est un tableau à une dimension alloué dynamiquement. Chaque position dans ce tableau représente une case de la grille et contient l'ID de la tuile qui s'y trouve.
- **empty_index (int)** : Cet entier stocke l'index précis de la case vide dans le tableau permutation. Cela permet de vérifier très rapidement si un mouvement est valide.
- **Moves (unsigned long)** : C'est le compteur qui enregistre le nombre total de déplacements effectués par le joueur.
- **is_solved (int)** : Un simple indicateur qui vaut 1 si la partie est gagnée, et 0 sinon.
- **selected_image_index (int)** : Stocke l'index 0, 1 ou 2 correspondant à l'image choisie par le joueur dans le menu.
- **all_tiles** : Un tableau de pointeurs vers les structures tile_t. Ces structures contiennent toutes les informations visuelles nécessaires à chaque tuile (chemin de l'image, coordonnées de découpage, largeur et hauteur).

4.2 Conventions de Représentation

Pour la logique du Taquin, nous avons utilisé des conventions précises dans le tableau permutation :

- **Tuiles Pleines** : Elles sont identifiées par des IDs
- **Case Vide (La Tuile qui manque)** : Elle est identifiée par l'ID -1.
- **Condition de Victoire** : La fonction game_is_solved renvoie Vrai uniquement lorsque l'ID -1 est situé à l'index 0 du tableau, et que les IDs des autres tuiles se suivent dans l'ordre croissant.

5. Algorithme de Mélange et Preuve de Solvabilité

5.1 Méthode Utilisée

Pour garantir que le joueur puisse toujours gagner, nous n'utilisons pas le mélange par permutation aléatoire (qui pourrait créer un état non résoluble) mais on utilise plutôt la méthode du Random Walk :

- **Fonction** : `game_shuffle_safe(state, 500)`.
- **Mécanisme** : Nous partons de l'état résolu (solvable par définition) et nous appliquons une séquence de 500 mouvements valides aléatoires (effectués par `game_move_tile`). Le compteur de coups est remis à zéro à la fin de ce processus.

5.2 Démonstration de Solvabilité

- **Principe de Preuve** : Le Taquin est basé sur le concept de parité (une configuration est solvable si elle a une parité identique à l'état résolu).
- **Démonstration** : Chaque étape de notre algorithme (`game_move_tile`) effectue un mouvement valide. Comme l'état initial est l'état résolu, et que chaque déplacement est l'inverse d'un déplacement possible, toute configuration obtenue par cette séquence est, par définition, atteignable.
- **Conclusion** : La configuration finale obtenue par l'exécution de `game_shuffle_safe` peut toujours être résolue par le joueur en appliquant la séquence inverse des mouvements qui ont été exécutés.

6. Conclusion Personnelle

Ce projet Taquin était vraiment un gros défi technique pour nous. L'exigence de développer une application graphique complexe en C89 mais tout en gérant les contraintes de la librairie de l'IUT a rendu le travail très difficile par rapport à notre niveau actuel.

Malgré ça nous avons vraiment apprécié le défi et cela nous a forcés à nous entraîner et à effectuer des recherches très approfondies. Nous regrettons malheureusement de nous être pris un peu tardivement, mais cela nous a contraints à plonger directement dans des concepts du C que nous ne maîtrisions pas complètement. Nous nous sommes donc améliorés sur des points importants du langage comme l'allocation dynamique de la mémoire, la gestion des tableaux pour représenter des grilles de taille variable ou aussi l'importance de la modularité et de la séparation des préoccupations.

Nous avons vraiment aimé travailler sur la complexité de l'algorithme de mélange garanti solvable et la nécessité de l'implémenter rigoureusement pour la validation. Malgré la latence et les difficultés de débogage liées aux problèmes du serveur graphique X c'était enrichissant. Nous sommes assez satisfaits du résultat final, même si nous savons qu'il aurait pu être encore mieux optimisé.