Préambule :

Ce document servira de compte rendu pour le projet d’algorithmie. Il y sera renseigné une partie de l’organisation, de la reflexion et de l’avancement du projet et les problème rencontrés. Ce document est decoupé en 3 parties : l’Organisation, la conception et le devellopemment.

L’organisation concernera la gestion du projet et les methods de travails. La conception sur sur la reflexion sur la manière de coder le programme. Et le devellopement sur la résolution des problèmes

Organisation :

Pour pallier a la mauvaise organisation du précedent projet nous nous somme donc imposé un planning de projet ( Planificateur de projet.xlsx ) ainsi que 3 réunions durant la semaine a des heure fixe.

Comme pour le précedent projet nous commenceront par réalisé un jeu de base simple auquel nous implementerons avec le temps de nouvelles fonctionnalités.

Nous nous attarderons sur la logique du programme avant de faire sont aspect graphique.

Chaque function et morceau de code sea commenter pour permettre un meilleur comprehension du code au sein de l’équipe.

La hiérarchie du projet est organiser tel que :

* « src/ » : est le dossier dans lequel le code source est situé
* « src/ressources/ » : est le dossier ou sont situé les fichiers ressource
* « test/ » : est le dossier dans lequel sont placés les test de vérification
* « build/ »

Beta test par les frères et soeurs

Conception :

La grille de jeu sera surement representer par un tableau d’entier ou d’une nouvelle structure à double dimension .

Une nouvelle enum representera les direction TOP,RIGHT,DOWN, LEFT

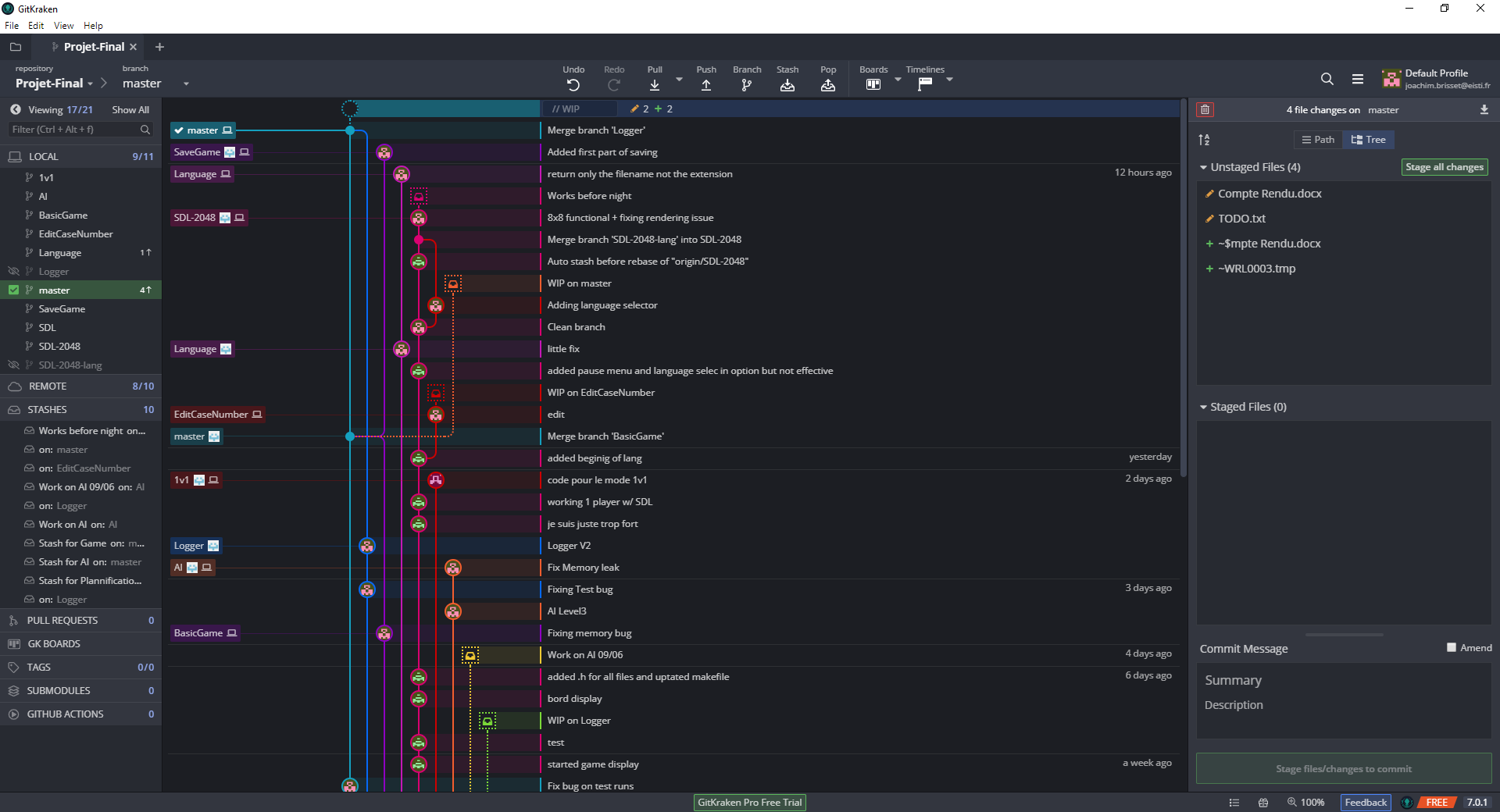
La logique du jeu sera representer par un fonction qui initialisera le jeu au depart puis suivi d’une boucle infinie qui genere une nouvelle case affiche le plateau et attend un mouvement. Cette boucle se terminera lorsque le jeu est finis et enfin suivi pas des instructions qui termineront le jeu et liberront la mémoire.

Une fonction qui deplace et fusion tout les boite sur une direction

Une fonction qui verifie qui tout les case sont plein pour savoir si on peut generer une nouvelle case

Une fonction qui generera une nouvelle case avec sont score

Développement

Repo : La première étape fut de créer un repository sur GitHub pour facilité la collaboration au sein du projet. Plus besoin de s’envoyer les différents fichiers, pas de problème lors d’une édition simultanée. Pour faciliter la vie de certain qui n’avait pas le courage d’apprendre le bash de git nous avons utilisé un programme nommé GitKraken très simple d’utilisation :

Makefile : Le *makefile* fut un des premiers développements du projet. Tous au long du projet il a été améliorer pour facilité le développement du projet. Son utilité principale est de compilé et copié tous les fichiers nécessaires au bon fonctionnement du programme ainsi que l’exécute sans problème. Actuellement il faut être dans le dossier build/bin/ pour l’exécuter sinon le programme ne trouve pas les fichiers additionnels à charger. On peut résoudre ce problème en utilisant le 1er argument passé à l’exécution de la fonctions main.

Additionnement le makefile permet de lancer et compilé des tests sur les fonctionnalités de manière modulaire : make test (TEST=<testfile>). Il peut aussi compresser tout le projet simplement a l’aide de la commande : make zip

Pour une liste détaillée des sous-commandes : make

Gestion des Headers : Tous fichiers source c sont accompagné d’un fichier header associé dans lequel l’ensemble des fonctions sont prototyper et les macros rassemblées. De plus il y sera rajouté une sécurité pour ne pas inclure plusieurs fois les fichier :

Exemple :

#ifndef EXAMPLE // if not define do

#define EXAMPLE

#include <library> // first include classic library

#include “file” // then include non classic library

typedef struct {}; // next declare structs and enum

#define DEFINE 10 // next declare all define

extern int GlobalVar // declare all global variable as extern or static

int function(); // finally declare all function prototype

#endif // end if

Basic Game : La première étape du développement du jeu fut les structures de base qui seront utilisé par le jeu. Elles seront stocké dans header séparé, GameStructs.h, pour être facilement accessible par tous les autres fichiers.

Une case du jeu sera représentée par la structure Box :

typedef struct {

....int value; /\*< the score of the box\*/

} Box;

L’utilisation de cette structure permettra une implémentation rapide des nouvelles fonctionnalités sur les cases du 2048 (Ex : Diffèrent type de case).

Le jeu sera représenté par une nouvelle structure qui contiendra toutes les informations nécessaires au bon fonctionnement de ce dernier.

typedef Board Box \*\*\*;

typedef struct {

....int size; /\*< size of the board\*/

....Board board; /\*< the game board \*/

} GameState;

Le type Board n’est pas un nouveau type il sert juste a rendre le code plus lisible et compréhensible. Cette structure centralisera les données du jeu et simplifiera leur passage en argument.

Logger : Cette fonctionnalité permet de mieux gérer les messages d’information du programme en leur attribuant un niveau d’importance et leur manière de se diffuser. Il permet aussi de retrouver les erreurs plus aisément grâce un formatage des messages avec diverses informations ajoutées et de désactiver facilement ces messages quand ils sont inutile.

int function() {

.... log(INFO, “this is a %s log”, “test”);

}

log() is a macro which replace this function by another containing the current function name and file:

#define log(level,...) logg(level, \_\_FUNCTION\_\_, \_\_FILE\_\_, \_\_LINE\_\_, \_\_VA\_ARGS\_\_) //not use log(level,format,...) because '...' must not be null

La fonction setLogger() permet de définit le comportement du logger : la méthode de diffusion les niveaux d’importance à ignorer ainsi que le fichier où enregistrer les messages.

Language : Cette fonctionnalité permet d’ajouter des traductions au jeux. Pour cela il suffit de rajouter un fichier .lang dans le dossier build/bin/ressources/lang/. Quand une langue est sélectionnée toutes les traductions sont chargé en mémoire dans une liste chainée de structure Translation avec un identifiant et une traduction.

Pour traduire il suffit d’appeler cette fonction avec l’identifiant :

getTranslatedText(“debug.test”);

Cette fonction va chercher dans la liste chainée l’élément qui le même identifiant et retourne sa traduction associée.

getAvailableLanguage();

Cette fonction permet de trouver toutes les traductions disponibles. Elle retourne une liste chainée de structure TranslationFile avec un champ pour le nom du fichier associer a la langue et un champ qui stockera la langue de traduction.

SaveGame : Cette fonctionnalité permet de sauvegarder une partie et de la charger après. (en cour)

AI : Cette fonctionnalité permet au jouer tout seul ou de jouer en 1v1 avec celle ci

Main.c : Ce programme est le « tronc principal » du jeu. En effet le jeu se déroule avec la même fenêtre et renderer. Les bibliothèques externes SDL\_ttf et SDL\_mixer ont été ajoutés pour respectivement pouvoir écrire sur la fenêtre et pouvoir gérer facilement la musique. C’est aussi dans ce main() qu’est initialisée notre fonction de « traduction » qui sera utilisée tout au long du jeu. Techniquement parlant, cette fonction représente le menu principal du jeu.

Option.c : fonction permettant de gérer le volume, le thème ainsi que la langue du jeu. LE volume est géré facilement car il est directement mis à jour pour toute la fonction. Le changement de thème modifie en fait des pointeurs vers code RGB qui va alors être utilisé dans tout le programme. La langue choisie pointe juste vers une structure TranslationFile qui contient le nom du fichier .lang.

Pause.c : fonction créant un menu pause qui stoppe temporairement le déroulement du jeu lorsque l’on appuie sur ‘esc’. Possibilité de reprendre la partie ou de tout simplement la quitter.

Gamover.c : fonction fondamentalement semblable à pause.c. S’affiche à la fin de la partie. Permet soit de revenir au menu principal soit de relancer une partie dans le même mode de jeu directement.

selectionTitle : fonction permettant de choisir son mode de jeu. Dans le cas du 4x4 et des 8x8 lances la fonction Game.c avec des paramètres différents