Remettre les carrés sur le graphique. Peut-être remettre les noms des répertoires sur le graph 3: data pour données du jeu, données générées par le jeu backup, fichier stdout pour le log, méthodes d'accès aux dossiers tools, structures de données (pokémon et field), structure de jeu dans battle et main, commandes graphiques dans Graphics

Idée: les structures de jeu modifient les données mais n'apportent pas d'info, les structures de données stockent les objets de façon hiérarchisée

Random: en fait n'était pas random, une des difficultés du code, à développer

Tout est dans un fichier où je vais aller chercher les informations concernant le type à partir du nom. Idée: fire est un objet de type string, et pas besoin d'utiliser type.GetName=fire

Dans Battle, il y a Player et Trainer (dont on ne s'est pas servis). Player contient les Pokémon. La méthode importante est SwapPokémon (ne pas parler de getpokémon). Parler de battlewildpokemon (2-3 pages) en détaillant son fonctionnement. Séparer les méthodes void et non void. tout est privé dans battle. (plein de petites fonctions s'y appellent mais le main ne peut pas y accéder).

Dans Graphics (contient toutes les fnc graphiques), presque aucune ne touche aux structures du jeu sauf celle qui touche à la vie (hpallydecrease et hpfoedecrease). quand tu envoies un pointeur d'un objet dans une classe, tu peux l'utiliser mais à l'extérieur il n'a pas bougé.  
2 natures de graphiques: Field et le reste, field pour la map et les autres pour le combat. Leur mécanisme est d'être des fonctions void, ou alors qui renvoient un flag avec un type. Soit ces flags sont demandés par l'utilisateur, typiquement entrée pour afficher un menu, par exemple le fight menu si l'utilisateur clique sur fight. Toutes ces fonctions prennent en paramètre l'écran et décident tour à tour sur l'écran, et on ne voit pas que les programmes sont différents (ex: passage menu global menu de combat) grâce au double buffering. Il existe les flags volontaires et involontaires (ex; rencontre avec un pokémon aléatoire, qui fait spontanément s'arrêter la machine). (Par exemple pour sauvegarder la position il faudrait renvoyer le flag et les 2 positions. Les flags sont un peu une levée d'erreurs, ici d'action spontanée)

Tous les arguments qu'on a mis: l'arrière plan (background), la base de l'ennemi, celle de l'allié, le pokémon ennemi, l'allié, la barre de menu en bas, les 2 databox (qui servent à récupérer des données: celles où on écrit la vie), le texte du nom, du level, l'image de la barre de la vie et la barre de la vie (tout ça pour les 2), et la vie sous forme d'entier pour l'allié.

Dans toutes les fnc de graphics (sauf battleh), on avait un pb: comment faire un jeu en évitant d'avoir une seule grosse fnc de 10k lignes? On est arrivés à cette solution là (exemple dans toutes ces fonctions, les 5 lignes set\_battle...)

on a utilisé un namespace pkmn pour toutes nos fonctions. graphics a été fait en c avec la commande extern c, qui permet au compilateur c++ de compiler les fichiers comme si c'était du c.

Un paragraphe avec toutes les données. Données extraites de pokémon essentials avec des algos python. données texte, images (retriées, ajustées à partir des images existantes), algo python pour découper en tiles. On a les fonts. backup fait des sauvegardes à partir de tables. tables est très bien pour la lecture mais plus délicat pour l'écriture. pour la fonction faisant l'effet visuel avec le cercle, il a fallu sauvegarder l'écran précédent. tu ne vois pas le bg mais l'écran que tu as sauvegardé en arrière plan pendant le fondu (pour ne pas avoir à le recréer). cette technique ne marche pas partout, pas dans battle par exemple parce que la vie change.

tous les chemins sont dans filepath.

on a du télécharger sdl, sdl image, sdl ttf (pour les fonts), (sans les dll l'algorithme ne tourne pas)

le main est une boucle infinie définissant l'écran, qui d'arrête grâce à un flag.

on a ajouté un fichier ressources pour avoir une icone plus jolie.

séparateur ; car on était en anglais (à rajouter tout en haut)

ce que fait le programme, comment le faire fonctionner, comment il est architecturé, pb rencontrés et solutions adaptées, description de l'architecture générale du programme.