Projet : le jeu mario 2D guide

Les pré requis nécessaire pour la réalisation de ce projet : entrée/sortie, condition, boucle, fonction, pointeur, tableau (à 2 dimension), structure, fichier, la gestion d'erreur, allocation dynamique, librairie SDL et Github.

Si vous souhaitez travailler depuis la maison sur ce projet, il vous faudra installer sdl2 et sdl2-image. Je vous conseille fortement de l'installer sur une distribution linux et non windows, c'est beaucoup plus simple. Si vous n'avez pas linux sur votre ordinateur, vous pouvez soit utiliser une machine virtuelle, soit vous pouvez utiliser le sous système linux pour windows WSL, pour se faire ouvrez l'application executez et écrivez "optionalfeatures". Cochez la case Sous-système Windows pour Linux puis faites OK, suivez ensuite les fenêtres qui s'affiche et redémarrez le PC. Une fois que vous avez WSL; vous pouvez installer une distribution Ubuntu via le microsoft store. Vous pouvez alors ouvrir une application Ubuntu (avec le numéro de la version) LTS qui vous donnera accès à une console Linux classique (sans interface graphique), vous pouvez aussi remarquer un nouveau dossier sur votre PC qui contiendra l'ensemble des dossiers de cette "partition" Linux. Depuis la console Linux, écrivez les 2 lignes de commande suivante :

- sudo apt-get install libsdl2-dev
- sudo apt-get install libsdl2-image-dev

Voila vous avez installé les 2 librairies nécessaire pour lancer le projet (bien plus facile que sur windows). Vous pouvez alors facilement compiler votre projet via les commandes classiques sur la console Linux, si vous voulez un peu plus de confort vous pouvez installer sur votre windows l'IDE suivant : visual studio code. Il vous permettra de coder depuis votre windows, et de sauvegarder les différents fichier sur votre partie Linux pour effectuer la compilation. Vous devrez juste installer les extensions suivantes (partie droite du logiciel, ou raccourcie : Ctrl + shift + X): C/C++ et remote development.

Il vous suffira alors d'appuyez sur la petite icone verte en bas à gauche (avec les 2 chevrons blanc), vous aurez alors une barre de recherche qui s'ouvre, sélectionnez "New WSL Windows using distro..." puis de sélectionner votre distribution de linux que vous avez installé. Vous pouvez alors facilement ouvrir un dossier à un endroit de votre "arborescence" Linux et y créer votre projet. Vous aurez alors à juste positionner votre console Linux aux bon endroit et de compiler/lancer votre code.

Si vous souhaitez quand même tout faire sur windows (même si la solution que je vous propose et assez proche), je vous conseille de regarder plusieurs tutorial sur l'installer de la bibliothèque SDL2 (et pas 1) et de comment l'utiliser avec un IDE.

Activite 1

Nous allons réaliser un jeu en 2D, le premier mario bros. Le but sera de réaliser un jeu en 2 dimensions en utilisant la librairie SDL, l'ensemble du code sera en C. L'objectif final, sera un jeu ou on peut diriger notre personnage dans les 2 directions (+saut), au travers d'un niveau. Plusieurs améliorations seront proposées pour rajouter des options qui amélioreront le gameplay. Vous trouverez ci dessous un exemple de jeu (voir figure 1).

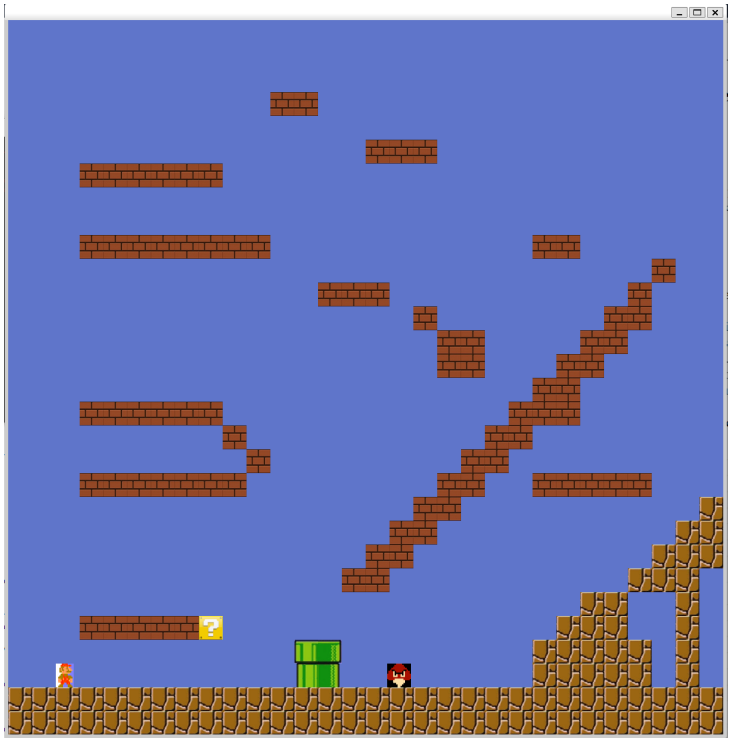

Figure 1: Jeu mario 2D

Vous allez réaliser ce jeu en binôme et pour partager votre code, vous utiliserez un logiciel de versioning qui est Github. Vous devrez gérer vous même les moments ou vous effectuerez vos commits et vous assurez que le code que vous envoyez à votre binôme est bien fonctionnel.

Vous aurez à disposition l'ensemble des sprites (mario avec les diverses direction, goomba, sol, ciel, bloc, tuyau, drapeau, bloc jaune), des fichiers qui vont représenter les niveaux et plusieurs fichiers représentant une architecture du logiciel.

Pour un projet plus important (comme celui-ci), il est important de diviser le code en plusieurs fichiers qui regrouperont des fonctions sur un thème similaire. Ici, nous auront les fichiers suivant :

- main.c : contient la page d'introduction et permet de lancer notre jeu, c'est le fichier qui va contenir la fonction main (et le seul fichier qui aura cette méthode). Vous pouvez voir la page d'introduction sur la figure 2 (il s'agit d'une simple image).
- game.c et game.h : contient le coeur du jeu, dont notamment la boucle qui va appeler toutes les fonctions pour faire tourner le jeu. Toutes les méthodes en question seront dans d'autres fichiers.
- file.c et file.h : contient les fonctions qui vont charger les fichiers images et map. L'affichage de la map sera aussi géré dans cette méthode.
- charactere.c et charactere.h : contient la création des personnages (mario et éventuellement les ennemis) et leurs affichages.
- event.c et event.h : gère l'ensemble des évènements du jeu, c'est à dire les mouvements, le scroll de la map, les collisions (avec les obstacles, les ennemis), la sortie du décors et la fin du jeu (perdre ou gagner).

PS: tous les fichiers .c hormis le main.c, auront un fichier du même nom mais en .h qui sera le fichier de header. Ce fichier contiendra les prototypes des fonctions du fichiers .c, les différentes structures nécessaire de notre fichier et éventuellement les include qui sont

nécessaire. Pour compiler cette ensemble de fichier vous aurez besoin de la ligne de commande suivante : **gcc main.c file.c game.c event.c charactere.c -o mario -lSDL2main -lSDL2_image -lSDL2**. Le début représente l'ensemble des fichiers que vous allez compiler (pas besoin de compiler tous les fichiers si vous ne les utilisez pas encore) et la fin représente l'ensemble des librairies nécessaire pour le projet.

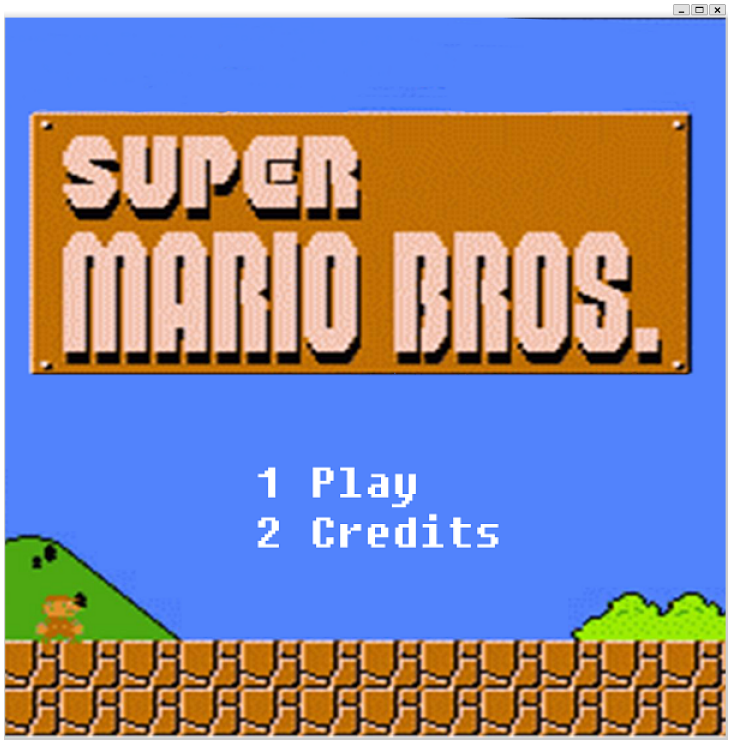

Figure 2: Fenêtre d'intro de mario

Pour vous aider vous aurez accès à différentes structures qui vous seront fournis pour vous aider lors de la programmation des différentes parties.

Attention, un soin particulier devra être mis en place sur la rédaction des commentaires dans votre code. En effet, vous allez coder à 2 personnes sur un même projet, votre collègue ne sera pas forcément au courant de ce que vous avez voulu faire avec un algorithme ou une certaines fonction, donc commentez votre pour votre et votre binôme. De plus il n'existe pas seulement qu'une seule bonne solution (il existe énormément de solution possible), donc ne vous arrêtez pas à ce que pourrez avoir codé quelqu'un de la classe. Finalement, un conseil très important pour la stabilité de votre code, à chaque fois que vous utilisez un malloc (pour instancier une structure, ...), il doit y avoir à la fin de votre code un free pour libérer la mémoire, de même pour la création de texture et sa libération avec SDL_DestroyTexture et pour le renderer/windows. Si vous oubliez de faire le ménage à la fin de votre code, vous risquez d'avoir des bugs très particulier et incompréhensible.

Pour commencez vous allez travailler ensemble sur l'écran d'introduction et la gestion du dépôt github en parallèle.

(a) Mettez en place le dépôt sur Github en le partageant entre vos 2 comptes. Effectuez le premier dépôt sur celui ci pour l'initialiser.

(b) Dans le fichier main.c, réalisez un code permettant de générer une fenêtre vide de taille 1200*1200 (utiliser les constantes LARGEUR_FENETRE et HAUTEUR_FENETRE de file.h).

(c) Rajoutez l'image "menu.jpg" qui va symboliser notre image d'introduction (voir figure 2). Ne pas oubliez de détruire la texture à la fin du programme (pour l'instant vous pouvez mettre un delay pour vérifier le bon fonctionnement).

(d) Récupérez les évènements du clavier pour rentrer dans le jeu. Vous devrez permettre de quitter la fenêtre si on clique sur la croix (bouton en haut à droite de la fenêtre) ou avec la touche echap (ne pas oublier de nettoyer le tout en quittant). Vous devez aussi permettre de lancer le jeu si on appuie sur la touche 1. Pour se faire, lors de l'appuie de la touche 1, vous lancerez la fonction jouer(SDL_Renderer* renderer) du fichier game.c.

(e) Finalement, rajouter un écran de crédits simple avec le nom de votre binôme qui sera accessible lorsque l'on appuie sur le touche 2 depuis le home screen.

(f) Mettez en place les 2 branches dans le dépôt Github, une branche pour la partie Map et une autre branche pour la partie personnage. Vous fusionnerez ces 2 branches quand vous aurez chacun fini les 2 parties.

Les 2 activités suivantes correspondent aux tâches a effectué par chacun des membres du binômes. Il faut bien avoir que le travail effectué par vous 2 est très connecté, et vous devrez discuter à plusieurs moments sur des points particuliers pour bien les appréhender. Si vous êtes bloqué à un endroit n'hésiter pas à regarder ce problème avec votre binôme. De même si vous avez fini votre parti, vous pouvez aider votre binôme sur la sienne pour pouvoir faire progresser le projet ensemble.

Activite 2

Cette partie se concentre sur la génération de la map et sur l'écran d'introduction.

Les questions qui suivent vont se réaliser dans les fichiers file.c pour la gestion des fichiers de niveau, et dans le fichier game.c pour utiliser les différentes méthodes réalisé dans le fichier file.c. Avant de commencer, vous devez comprendre plusieurs choses. Tout d'abord le niveau est stocké dans un fichier niveau0.lvl et niveau1.lvl, vous utiliserez dans un premier temps le niveau0 qui représente une plus petite fenêtre (et donc ne nécessite pas de scroll), vous pourrez utiliser niveau1.lvl une fois que vous aurez mise en place la gestion du scroll.

Vous pouvez ouvrir les fichiers de niveau, il s'agit seulement de tableau à 2 dimensions qui contiennent des chiffres. La première ligne du fichier représente son nom, la deuxième le nombre de case de notre niveau (largeur hauteur) et ensuite l'ensemble des chiffres représentant le niveau. Chaque chiffre va représenter une case de notre map et chaque case sera de taille 40*40 pixel (utiliser la constante Size_Sprite de file.h). Il faudra charger les sprites (images utiliser pour notre map) correspondant aux chiffres en question :

- 0 : correspond au ciel et à l'image sky.png.
- 1 : correspond au sol et à l'image sol.png.
- 2 : correspond au bloc de pierre et à l'image block.png
- 3 : correspond au boîte avec un point d'interrogation et à l'image boite.png
- 4 : correspond à une partie d'un tuyau et à l'image tuyau1.png
- 5 : correspond à une partie d'un tuyau et à l'image tuyau2.png

- 6 : correspond à une partie d'un tuyau et à l'image tuyau3.png
- 7 : correspond à une partie d'un tuyau et à l'image tuyau4.png
- 8 : correspond au poteau du drapeau qui sera utilisé en fin de niveau et à l'image fin1.png
- 9 : correspond au drapeau qui sera utilisé en fin de niveau et à l'image fin2.png
- 10 : correspond au lieu d'apparition des goombas, ils apparaîtront à cette endroit et bougeront pour le reste de la partie à partir de ce point.

Les images fournit ne sont pas définitive et si vous souhaitez les améliorer vous êtes libre de réaliser/récupérer d'autre image, essayer juste de rester sur la même taille d'image (40*40 pixel) pour éviter les déformations et de changer le code pour s'adapter.

Pour finir, vous aurez accès à 2 structures : Map et Sprites. La structure Map va contenir 5 champs : int width, height (qui vont contenir le nombre de carré qu'il y aura dans notre niveau), int **loadedMap (qui va contenir les chiffres du niveau qui sont dans le fichier) et int xscroll, yscroll (va contenir de combien vous avez scrollé dans les 2 directions pour faire bouger la fenêtre mais garder le personnage dedans). Cette structure contient donc toutes les informations nécessaire à la gestion de notre map et nous aurons besoin que d'une seule instance de cette structure. La structure Sprites quant à elle va contenir 2 champs : SDL_Texture* sprite (qui va contenir une image d'un sprite de notre map) et int traverser (qui va contenir l'information si ce sprite est traversable (0) ou non (1)). Vous aurez donc besoin d'une liste de taille 10 (vous pouvez utiliser la constante NbSprites de file.h) qui va contenir les 10 textures possible pour notre map, une seule instance de cette variable sera nécessaire aussi.

Bien entendu l'ensemble des champs et des valeurs que je donne ici ne sont que à titre indicatif, si vous avez des meilleurs idées de fonctionnement, vous pourrez les mettre en place si vous le souhaitez. De plus vous n'êtes pas obligé (et c'est même conseillé) de ne pas remplir et gérer l'ensemble des champs de ces structures dès le début, vous pouvez vous focalisé dans un premier temps à certains champs, puis à chaque fois que vous rajouter une fonctionnalité vous pourrez alors gérer un champ de plus.

(a) Programmez une fonction permettant de lire le fichier de niveau (niveau0.lvl) et de récupérer les informations de largeur et hauteur de la map, attention vous devez sauter la première ligne (vous pouvez la lire sans la stocker).

(b) Continuez la fonction précédente pour créer une instance de notre structure Map et remplissez le champs width et height. Il vous est conseille de créer avec une variable de cette structure avec un malloc dans la fonction jouer() du fichier game.c et de la donner en paramètre a votre fonction.

(c) Permettez à la fonction de pouvoir lire le tableau a 2 dimensions du fichier et de remplir le champs loadedMap de notre variable de type Map.

(d) Réalisez une fonction qui va remplir une liste de variable de type Sprites contenant l'ensemble des images de notre décors. Cette fonction va devoir ouvrir l'ensemble des fichiers images et préciser si chaque Sprites est traversant ou non. Il vous est conseille de créer avec un malloc cette structure dans la fonction jouer() du fichier game.c et de la donner en paramètre a votre fonction. Indice : l'ordre de cette liste à son importance, suivez les chiffres correspondant sur les fichier de niveau.

(e) Écrivez une fonction libererMap(...) permettant de faire le nettoyage en détruisant en fin de programme l'ensemble des textures et de libérer la mémoire non nécessaire.

(f) Utilisez les 3 fonctions que vous avez réalisé dans la fonction jouer() du fichier game.c pour créer notre Map et notre liste de Sprites. Vérifiez le contenu de notre Map avec un printf. Rajoutez la possibilité de lire les évènements clavier et notamment le bouton croix de notre fenêtre et la touche echap pour pouvoir quitter la boucle de notre programme.

(g) Programmez une fonction affichageMap qui permettra d'afficher la map sur notre fenêtre, elle devra prendre en paramètre le renderer, la liste de Sprites et la Map. Ne prenez pas en compte le numéro 10 de la Map (le goomba) et affichez le comme étant le ciel (vous changerez ça plus tard quand vous mettrez en place le goomba). Indice : il vous suffit de récupérer le numéro sur la map de loadedmap, de l'utiliser dans la liste de sprite et de placer l'image au bon endroit.

Activite 3

Cette partie sera plus focalisé sur la création de notre personnage mario, son mouvement et la gestion des obstacles. Pour se faire, vu que vous n'aurez pas accès dans un premier temps à la map, vous allez utilisez un écran blanc simple dans lequel vous réaliserez vos tests pour voir la bonne gestion de vos texture et mouvement.

(a) Créez dans le fichier game.c une fenêtre blanche permettant de jouer, réalisez le tout dans une fonction à part de la fonction main().

Les questions qui suivent vont se réaliser dans les fichiers charactere.c et event.c pour la gestion des personnages et des évènements et dans le fichier game.c pour l'utilisation des diverses méthodes au sein de la méthode jouer(). Tout d'abord il faut comprendre le fonctionnement et le déplacement de notre personnage (mario ici). Celui peut être assimilé à un rectangle sur lequel on colle une image qu'on déplace dans les 2 directions possibles (x et y). Ce rectangle va se déplacer dans la map qui sera géré par votre binôme mais ne pourra pas rentrer dans des éléments non traversable (voir structure Map). Pour finir vous aurez une structure a utilisé qui symbolisera notre personnage. Dans le fichier charactere.h, vous trouverez la structure Personnage qui contiendra 9 champs :

- SDL_Texture **image : une liste des images (de pointeur sur image pour être précis) qui contiendra l'ensemble des sprites de notre personnages (ici mario).
- SDL_Rect position : la position en l'instant t de notre personnage mario et plus particulièrement du carré qui le symbolise.
- int jump : symbolise si le personnage saute ou non, vaudra 1 à partir du moment ou le personnage commence à sauter et vaudra 0 à partir du moment où le personnage redescend (ou si le personnage ne saute pas).
- int jumptime : correspond à la durée depuis que le personnage effectue un saut, jusqu'au moment ou il atteint le point culminant de son saut. Après le point culminant atteint sa valeur va descendre jusqu'à retomber sur le sol. Son utilité principale et de bien gérer le temps de saut en mettant en place un temps de saut maximum que vous devrez trouver vous même.
- gravite : vaut 1 si le personnage est dans les airs (et donc que la gravité doit s'appliquer), vaut 0 sinon. Ce champ est utilisé pour tomber si il n'y a pas de sol en dessous du personnage.

- int direction et int dernierеDirection : symbolise la direction (ou la direction précédente) vers laquelle mario se dirige, vaut 1 pour la droite, 2 pour la gauche et 0 pour une absence de mouvement.
- int temps : doit être incrémenté de manière régulière et sera utilisé pour faire varier les sprites de mario pour donner une impression de mouvement.
- int win : symbolise plus l'état de la partie, vaut -1 si le personnage perds (tombe dans le vide par exemple), 1 si il gagne en touchant le drapeau et 0 sinon.

Bien entendu l'ensemble des champs et des valeurs que je donne ici ne sont que à titre indicatif, si vous avez des meilleurs idées de fonctionnement, vous pourrez les mettre en place si vous le souhaitez. De plus vous n'êtes pas obligé (et c'est même conseillé) de ne pas remplir et gérer l'ensemble des champs de cette structure dès le début, vous pouvez vous focalisé dans un premier temps à certains champs, puis à chaque fois que vous rajouter une fonctionnalité vous pourrez alors gérer un champ de plus.

(b) En utilisant la structure Personnage décrite ci dessus, réalisez une fonction permettant de créer une instance Personnage qui correspondra à Mario. Il vous est conseille de créer une variable de cette structure avec un malloc dans la fonction jouer() du fichier game.c et de la donner en paramètre a votre fonction. Remplissez les différents champs de notre structure avec des valeurs correspondant a la situation initiale (position initiale du personnage, direction, ...). Utilisez cette fonction (et toutes les suivantes) dans la fonction jouer() du fichier game.c.

(c) Rajoutez une fonction freePersonnage() permettant de libérer la mémoire des différentes allocation dynamiques effectué sur ce personnage et détruire les textures.

(d) Écrivez une fonction permettant d'afficher notre personnage sur notre écran dans son rectangle position, dans le fichier charactere.c, et affichez le résultat sur notre écran blanc (n'oubliez pas d'utiliser la fonction freePersonnage() en rajoutant un delay avant son emploie pour bien voir le résultat et bien gérer la mémoire). Vous ne prendrez pas en compte le changement de sprite pour l'instant, et vous utiliserez une seule sprite basique pour mario. Le code sera modifié plusieurs fois pour accomoder des fonctionnalités supplémentaire.

(e) Programmez une fonction permettant de récupérer les 3 directions de déplacement pour notre personnage (droite, gauche, haut pour le saut).

(f) Programmez une fonction de déplacement qui sera écrite dans le fichier event.c et utilisé avec la fonction précédente dans le fichier game.c. Cette fonction permettra dans un premier temps d'effectuer le déplacement à droite et à gauche de votre personnage sans changer les sprites de notre personnage pour l'instant. Vous testerez le tout sur votre écran blanc et ajusterez la vitesse de déplacement. Indice : il vous suffira de déplacer le rectangle du personnage qui sera alors raffraichi sur l'écran par le biais de la fonction d'affichage du personnage.

(g) Mettez en place le saut pour le personnage en utilisant les 2 champs de notre structure (jump et jumptime). Pour l'instant vous gérerez seulement le saut simple (et non la chute sans saut), c'est à dire que si le personnage saute d'un point A sans bouger, il doit revenir sur ce point A. N'oubliez pas d'incrementer/decrementez la valeur de jumptime a chaque deplacement. Testez la fonctionnalité pour trouver un bon équilibre pour la taille de saut et la vitesse de monté/descente.

(h) Complétez la fonction de déplacement (avec éventuellement une fonction de plus) pour changer les sprites en fonction du déplacement : droite, gauche, saut vers la droite et saut vers la gauche. Finalement alterner entre les sprites pour les mouvements sans saut entre la sprite statique et la sprite de mouvement (ex: Mario1.png et Mario2.png) en utilisant le champ temps de notre personnage et en l'incrémentant au bon moment. Testez le résultat pour ajuster la vitesse d'alternance des sprites.

Activite 4

A partir de maintenant, vous aurez besoin du code des 2 personnes pour effectuer les tâches suivantes. Vous pouvez néanmoins commencer à effectuer certaines de ces tâches. Dans le cas ou vous avez tous les 2 finis vos activités vous pouvez prendre des tâches différentes pour cette activité. Une répartition des tâches possible serait de laisser une personne gérer le scrolling, tandis que l'autre personne gère les obstacles. Une bonne partie des tâches qui suivent seront réalisé dans le fichier event.c, bien entendu après que vous aurez réalisé chacune des tâches, vous devrez l'intégrer dans la méthode jouer() du fichier game.c et la tester.

(a) Effectuez l'intégration du code entre les 2 activités précédente pour pouvoir effectuer le mouvement du personnage dans la map (il va bien entendu traverser tous les blocs).

(b) Programmez une fonction permettant de détecter la collision entre notre personnage (plus particulièrement son rectangle) et tous les éléments du décors autour de lui. Le but sera d'analyser dans la variable map tous les éléments présents autour de la position de notre personnage, de vérifier leurs valeurs et donc à quelle sprite elles correspondent pour finalement savoir si elles sont traversable ou non. En fonction la fonction doit renvoyer 1 ou 0 pour préciser si le personnage est en collision avec un objet non traversable ou non.

(c) Mettez en place la méthode précédemment faite en modifiant la gestion du déplacement pour stocker les déplacements effectué dans une variable de type SDL_Rect temporaire, vous éditerez alors le rectangle de position de notre personnage que si il n'y a pas de collision.

(d) Réalisez une fonction permettant de savoir si il y a un sol ou non en dessous du personnage. Vous pouvez vous inspirer du fonctionnement de la fonction qui gère les collisions.

(e) Utilisez la méthode réalisé précédemment pour et le champ "gravité" de notre variable Personnage pour faire tomber le personnage au sol si il n'y a pas de sol en dessous de lui.

(f) Programmez une fonction permettant d'empêcher mario de sortir de l'écran (à gauche, à droite et en haut seulement). Vous devrez calculer sa position par rapport à la map et vérifiez qu'il n'en sorte pas. De la même manière que pour les fonctions précédentes vous devrez intégrer cette fonction au sein du déplacement.

Le but des 3 questions qui suivent va être de mettre en place un scroll pour la map, c'est à dire que lorsque notre personnage bouge dans une direction, l'affichage de la map doit le suivre. On va suivre la règle suivante, dès lors que le personnage atteints la moitié de l'écran en largeur (en partant de la gauche), l'écran doit suivre le mouvement du personnage de tel sorte que le personnage doit rester au milieu de la fenêtre. Il se passe la même chose si le personnage arrive à la moitié de l'écran en hauteur. Pour se faire nous n'allons pas changer la

valeur de la position du personnage, mais modifié l'endroit ou on affiche le personnage sur la map et faire défiler l'affichage de la map.

(g) Mettez en place une fonction qui va remplir les champs xscroll et yscoll de notre variable Map. Pour se faire vous devez considérer 2 cas, soit le personnage n'est pas assez loin sur l'écran et dans ce cas les 2 champs valent 0, sinon ils doivent être égale de combien le personnage aurait bougé depuis le centre de l'écran.

(h) Modifiez la/les fonctions d'affichage de map pour prendre en compte ce scrolling. Ces modifications doivent permettre de faire défiler la map. Vous devrez agir sur la partie du champ LoadedMap de la variable map vous allez lire.

(i) Modifiez la/les fonctions d'affichage de personnage pour prendre en compte ce scrolling. Ces modifications doivent permettre de fixer le personnage sur le centre de la map lorsque l'on scroll.

(j) Modifiez vos fonction pour changer la valeur du champ win de notre personnage à 1 si on gagne (on touche le drapeau) ou -1 si on perd (on tombe dans un trou).

(k) Programmez une fonction permettant de gérer la fin de niveau. Elle doit afficher les images win.png ou lose.png (vous pouvez les modifier si vous voulez) dans la bonne situation. Vous devez alors permettre de quitter le jeu en appuyant sur la touche echap, ou de reprendre une partie en appuyant sur la touche 1.

Activite 5

Cette partie sera plus focalisé sur un ensemble de fonctionnalité additionnelle permettant une meilleure expérience de jeu.

(a) En utilisant la structure Personnage (mais pas besoin de l'ensemble des champs) et en vous inspirant des fonctions que vous avez utilisez pour mario, mettez en place l'affichage d'un goomba.

(b) Faites bouger les goombas de droite à gauche, en faisant changer la direction à chaque fois qu'ils percutent un mur.

(c) Programmez des fonctions pour gérer la collision avec les goombas, en cas de collision frontale avec un goomba, cela doit entraîner une défaite et si le personnage saute dessus, mario doit rebondir et faire disparaître le goomba (sprite 3 du goomba).

(d) Rajoutez un chronomètre en haut à droite de l'écran, démarrant à 2 minutes et diminuant au fur et à mesure. Lorsque le chronomètre atteint 0, le joueur a perdu.

(e) Rajoutez un score juste en dessous du chronomètre. Le score sera calculé de la façon suivante (plusieurs points seront implémenté plus tard) : casser un bloc rapporte 5 points, chaque pièce rapporte 15 points, tuer un goomba rapporte 10 points. A la fin du niveau, en fonction de la position ou mario touche le drapeau, il gagne plus ou moins de points (100 points tout en haut, 10 points en bas). Finalement, à la fin de la partie le temps restant est convertit en points (1 seconde = 1 point).

(f) Rajoutez des pièces dans les blocs avec un point d'interrogation. Lors d'un saut de mario sous un bloc avec un point d'interrogation, un pièce doit apparaître au dessus. Cette pièce sera immobile et doit disparaître après contact avec mario. Le nombre de point doit alors changer. Attention, mario ne peut pas re utiliser un bloc jaune qui a déjà été utilisé (indice : vous pouvez utiliser une deuxième variables tableau dans la structure map qui contiendra les cases utilisées).

(g) Mettez en place le système de champignon dans mario. Ils doivent apparaître de manière aléatoire sur un bloc (de la même façon qu'une pièce) avec une probabilité de 50% par rapport aux pièces, lors de leurs apparitions, ils doivent partir dans une direction aléatoire et bouger de la même façon que les goombas (rebondir de droite à gauche sur les murs).

(h) Programmez la collision de mario avec un champignon, lors de cette collision mario doit gagner une vie et changer de sprite d'affichage (voir image de mario non utilisé). La vie sera perdu lorsque mario touche un goomba et il aura de nouveau les sprites classiques. De plus mario peut casser les blocs en sautant en dessous d'eux dans cette forme (et faire des points supplémentaire).

(i) Rajoutez des musiques dans le jeu (voir les fichiers musiques fournit), vous avez à disposition une musique d'ambiance, pour le saut, en cas de défaite/victoire, pour le champignon, pour cassé un bloc, pour les pièces et lorsque vous tuez un goomba.

(j) Mettez en place d'autres ennemies comme les koopas.

(k) Proposez des améliorations du jeu.