UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

ISTY

École d'ingénieurs

RAPPORT DE PROJET

FINAL

Projet Super Bomberman

Réalisé par MASTOUR Mustapha & YAKDI Zakaria

2017

ISTY – CPI 2

SOMMAIRE

INTRODUCTION

Présentation du projet

Le projet informatique de deuxième année de cycle préparatoire est très important à nos yeux. Ce projet consiste à utiliser toutes les connaissances acquises en C durant le cycle préparatoire (et plus encore) pour réaliser quelque chose de concret. Contrairement à la plupart de nos camarades, notre choix ne s'est pas porté sur la stéganographie. Ayant tous les deux des connaissances en SDL 1.2, nous avons décidé de choisir le projet suivant : Reprogrammer le jeu vidéo Super Bomberman. Nous avons choisi ce projet pour deux raisons : D'une part, nous avions tous les deux pour souhait de créer notre propre jeu vidéo et d'autre part, ce projet est idéal pour mettre en pratique nos connaissances en SDL.

L'objectif principal de notre projet est donc de créer un jeu ayant les mêmes caractéristiques que le jeu Super Bomberman. C'est-à-dire faire déplacer plusieurs personnages dans une carte et les faire s'affronter en lançant des bombes pour détruire les obstacles et éliminer les adversaires.

Nous avions également pour objectif secondaire de créer une intelligence artificielle. Autrement dit, pouvoir jouer seul contre un adversaire autonome et intelligent.

Nous verrons dans une première partie les étapes de développement, en expliquant comment fonctionne notre code, puis dans un second temps, nous ferons un bilan des objectifs, en indiquant les objectifs atteint ou non atteint.

Etat du projet en Janvier

Lors de la dernière soutenance qui avait lieu le Vendredi 13 Janvier, notre projet avait déjà bien avancé. En effet, nous avions déjà une carte, un personnage qui pouvait s'y déplacer ainsi que poser des bombes (qui n'explosaient pas). Cependant, ce jour-là, le jury nous a fait remarquer que notre code avait quelques défauts. Tout d'abord notre personnage qui se déplaçait case par case nous éloignait de notre objectif qui était de se rapprocher le plus possible du véritable jeu vidéo ou encore que notre code n'était pas adapté pour intégrer d'autres personnages ainsi qu'une fonctionnalité phare du jeu vidéo : ajouter des bonus à nos personnages.

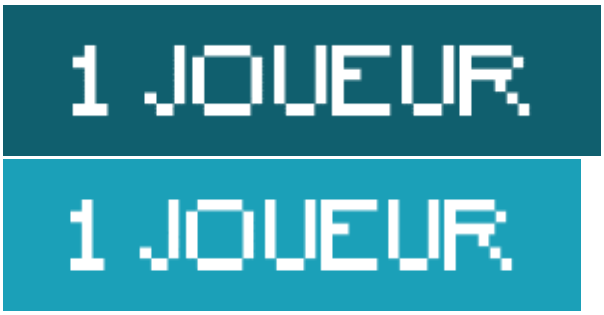
Suite à cela, nous avons décidez d'améliorer notre code afin de pouvoir facilement anticiper les fonctionnalités à venir.

ETAPES DE DEVELOPPEMENT

Menu

Nous considérons cette partie comme étant la plus simple du projet. La conception du menu consiste à créer deux surfaces différentes pour chaque bouton présent dans le menu (Mode 2 joueurs, 3 joueurs, 4 joueurs ou Quitter). Parmi ces boutons on pouvait distinguer deux images différentes : l'une lorsque le bouton est sélectionné et l'autre lorsqu'elle ne l'est pas.

EXEMPLE :



L'image du bouton sélectionné étant celle de droite et celle du bouton non sélectionné celle de gauche.

Pour pouvoir passer d'un mode à un autre, nous avons créé une variable « choix » qui prend les valeurs 2, 3, 4 ou 5. Cette valeur change lorsque l'on appuie sur une touche directionnelle. Il a donc fallu utiliser des macro-constantes de la SDL permettant de gérer les touches du clavier (SDLK_UP et SDLK_DOWN). Ces valeurs étant associées aux images des boutons sélectionnés, cela donnait l'illusion de passer d'un bouton à un autre. La fonction appelant le menu nous renvoi la valeur de « choix ». Il suffisait d'inclure cette valeur dans une condition « if » pour lancer le jeu avec le nombre de joueurs demandés grâce à la fonction « jouerPartie » qui prend en paramètre le nombre de joueur.

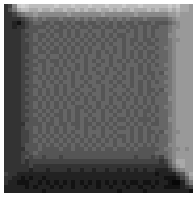
Carte

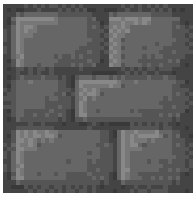
La conception de la carte se divise en deux parties : La première en console et la deuxième dans une fenêtre avec SDL 1.2. La carte est représentée dans un fichier texte sous forme de tableau à 2 dimensions. Chaque chiffre représente une icône (1 pour un mur, 2 pour une brique, etc…). On récupère ensuite notre carte dans nos fichiers sources à l'aide d'un tableau à 2 dimensions en lisant chaque caractère du fichier un par un et en l'inscrivant dans un tableau 2D. On fait parcourir une variable de position sur chaque case de la carte et on inscrit, pour chaque case, la surface correspondante au chiffre situé sur cette case. Le résultat est ensuite visible sur la fenêtre de jeu.

Durant une partie, nous serons amenés à modifier cette carte de jeu selon les actions du joueur. On pourra voir apparaitre des bombes, des flammes, et des items bonus récupérables par le joueur.

Pour finir, les personnages ne sont pas présents dans la carte. C'est-à-dire que le joueur est indépendant par rapport à la carte.

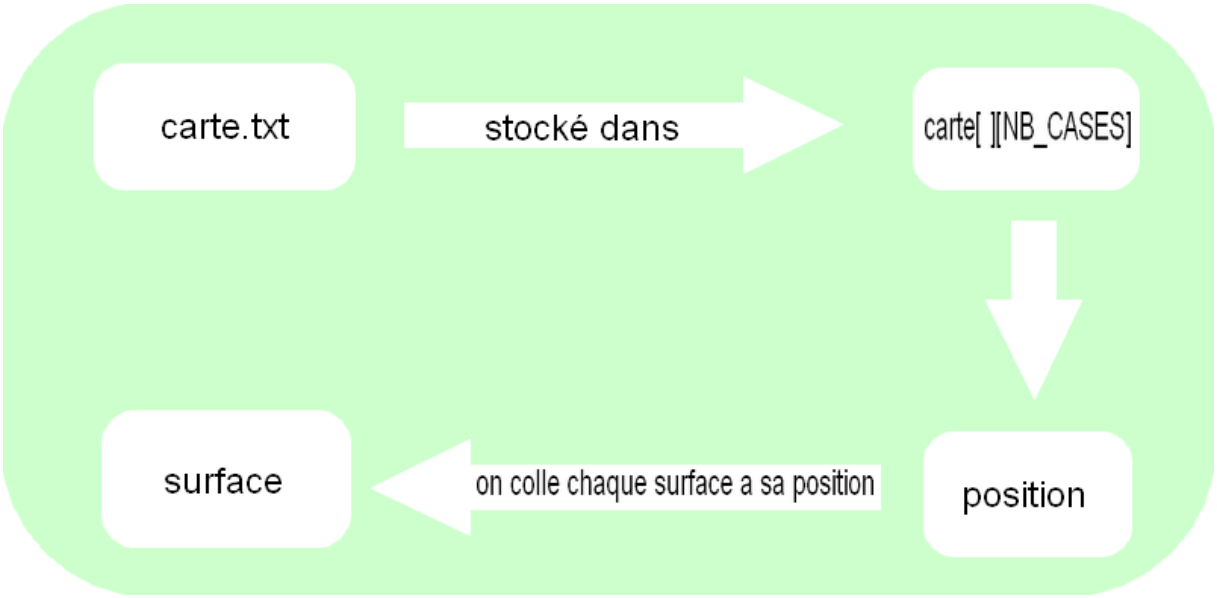
VOICI LES SPRITES QUE NOUS UTILISONS POUR CREER LA CARTE :





La première image représente un mur et la deuxième une brique. Nous avons décidé de ne pas mettre d'image pour représenter le sol, l'arrière-plan étant suffisant.

RECAPITULONS :



Personnages

Pour la création des personnages, nous avons choisi d'utiliser une liste chaînée qui répertorie chaque joueur. Tout d'abord le nombre de joueurs initialement présents dans une partie varie selon le choix de l'utilisateur (voir partie Menu) : il faut donc allouer une quantité de mémoire correspondant à ce choix. De plus, lorsqu'un joueur meurt au cours d'une partie, nous

voulons qu'il disparaisse non seulement de la carte mais également de la mémoire de l'ordinateur. D'où notre souhait d'utiliser des listes chaînées.

Chaque personnage doit avoir des caractéristiques propres à lui-même.

- Son numéro : c'est-à-dire savoir qui le joueur 1, 2, 3 ou 4.
- Son costume (ou « skin ») : chaque personnage doit disposer de son propre costume qui serait caractérisé par un sprite différent en fonction du costume.
- Sa position : position exprimée en pixels du coin supérieur gauche de la surface graphique du personnage
- Sa hitbox : position exprimée en pixels du coin supérieur gauche de la hitbox du personnage.
- Ses coins : abscisse et ordonnée, sur la carte de jeu, des quatre coins de la hitbox du personnage.
- Des booléens indicateurs : Tableau de booléens qui indique si chaque touche directionnelle est appuyée, booléen qui indique si la touche est appuyée ou encore un booléen qui indique si le joueur est assis sur la bombe
- Sa vitesse (qui peut être augmentée grâce à un item bonus)
- Son nombre total de bombes (qui peut être augmenté grâce à un item bonus)
- Son nombre de bombes restantes
- Sa puissance de bombe (qui peut être augmentée grâce à un item bonus)

SPRITES UTILISES POUR LE PERSONNAGE :

Intéressons-nous désormais aux déplacements des personnages. Nous voulons que le personnage se heurte contre les murs, les bombes et les briques. A part ça, il doit être libre de se déplacer face à toute autre icône. Pour cela, on va utiliser les positions (abscisse et ordonnée) des 4 coins du personnage sur la carte. A partir des coins du personnage, on définit des nouveaux coins situés à l'endroit où le joueur serait si on lui avait accordé le déplacement. Si ces nouveaux coins coïncident avec un mur, une bombe ou une brique sur la carte, alors le joueur ne peut pas avancer. Sinon, il lui est permis d'aller dans la direction considérée.

Afin d'obtenir graphiquement une illusion de relief durant une partie, on définit une hitbox qui entoure le personnage. Lorsqu'on va programmer les collisions, au lieu de travailler sur la surface graphique du personnage, on va travailler sur sa hitbox : une boîte rectangulaire invisible qui pourra, elle, se heurter contre les contraintes physiques. Ainsi, la surface graphique du personnage sera plus libre vis-à-vis des icônes du décor et on obtiendra l'illusion de relief.

Cas isolé à traiter : lorsque le joueur vient de poser une bombe, on veut qu'il ait la possibilité de sortir de la case où la bombe a été posée. Cela ne serait pas possible si on utilise le système évoqué précédemment. On va lui interdire d'avancer dans le cas suivant : lorsqu'il est assis sur une bombe, qu'il cherche à se déplacer vers une autre bombe (à l'endroit dans

lequel il cherche à se rendre) qui n'est pas située sur la même case que lui. Sinon, il peut se déplacer.

Lorsque le joueur se déplace :
- on incrémente les positions de la hitbox et des coins de la hitbox selon la direction considérée et selon la vitesse du personnage.
- on repositionne la surface graphique du personnage selon le déplacement de la hitbox.

Pour en finir avec les personnages, nous allons évoquer l'interaction entre les joueurs et l'environnement. En effet, il existe 2 types d'interactions du joueur avec l'environnement :
- Lorsqu'un joueur touche une flamme
- Lorsqu'un joueur prend un item bonus

Afin de minimiser le travail de calcul du processeur, nous avons pensé qu'il était nécessaire de vérifier les interactions des personnages avec l'environnement uniquement lorsque des flammes ou bien des items bonus sont présents sur la partie. En effet, si la liste chaînée de bombes explosées ET si la liste chaînée d'items sont vides, alors aucune interaction n'est possible, il n'y a pas besoin de réaliser le travail de vérification.

- Lorsque des flammes sont présentes sur le jeu, on vérifie si chaque joueur est présent sur une flamme. Pour cela, il suffit de vérifier si au moins un des quatre coins de la hitbox du joueur coïncide avec une flamme sur la carte. Si c'est le cas, le joueur est éliminé : on le supprime de la liste des joueurs et on libère la mémoire qui a été nécessaire pour stocker ses données. Il n'apparaîtra plus sur la carte et ne pourra plus appuyer sur aucune touche.

- Lorsque des items bonus sont présents sur la carte, on effectue le même mécanisme. Si au moins un des 4 coins du joueur coïncide avec un bonus, alors on attribue à ce joueur le bonus correspondant : augmentation de la vitesse pour la prise de roller, augmentation du nombre de bombes totales et du nombre de bombes restantes pour un bombe, augmentation de la puissance des bombes pour une flamme.

Bombes

Lorsque l'utilisateur appuie sur la touche qui lui permet de poser une bombe, on vérifie :
- Qu'il n'est pas assis sur une bombe
- Qu'il n'est pas à court de bombes

Si ces conditions sont vraies, on peut procéder à la pose de bombe. Il était également nécessaire de vérifier que la touche n'était pas enfoncée, afin de l'empêcher de poser plusieurs bombes à la fois sans lever la touche.
A partir de la hitbox du personnage, on définit la position (en pixels) de l'endroit où l'on veut poser la bombe pile entre les 2 genoux du personnage. On définit ensuite une variable qui transcrit la position (en pixels) de la bombe en un repère sur la carte. Grâce à ce repère on peut poser la bombe sur la carte. Enfin, on ajoute la bombe posée à la fin de la liste chaînée correspondante en inscrivant ses caractéristiques (instant où la bombe a été posée, puissance, position, joueur qui l'a posée). L'instant où elle a été posée s'obtient grâce à la

fonction SDL_GetTicks(). Elle renvoie le nombre de millisecondes qui se sont écoulées depuis le début du programme.

Avant d'exploser, une bombe doit rester présente sur le terrain pendant un certain temps, que l'on a défini dans le code source grâce à une macro-constante (environ 3 secondes). Il a été indispensable de créer une fonction verifierDelai() qui permet de déterminer si un délai donné en paramètre était dépassé, à partir d'un instant initial. Nous avons eu recours à la fonction SDL_GetTicks() pour obtenir l'instant actuel. Si la différence entre l'instant actuel et l'instant initial est supérieure ou égale au délai souhaité, alors le délai est dépassé. Dans le cas de notre jeu, l'instant initial représente l'instant où la bombe a été posée. Lorsqu'au moins une bombe est posée sur le terrain, le programme va vérifier le premier élément de la liste des bombes posées. Il n'est pas nécessaire de vérifier les autres bombes, car elles sont classées dans l'ordre croissant d'apparition sur la carte. La prochaine qui va exploser (et dont on doit vérifier le délai) est donc la première de la liste. Si le délai est dépassé, nous pouvons faire exploser le premier élément : il s'agit de transférer l'élément de la liste des bombes posées vers la liste des bombes explosées.

Ensuite avant d'afficher l'explosion de la bombe, il est impératif de déterminer la portée de la bombe dans ses directions. Pour cela, nous avons fait parcourir des variables de position dans les directions en partant de l'explosion. Pour chaque direction, on avance d'une case dans la carte et on vérifie l'entité présente sur la position considérée : si ce n'est ni un item bonus, ni un mur, ni une brique, et que la puissance de la bombe est suffisante, alors on peut incrémenter la portée de la bombe dans cette direction. On obtient un tableau qui nous donne la portée dans chaque direction, que l'on inscrit dans les caractéristiques de la bombe.

Au moment de la détermination de la portée, nous avons dû préparer en amont les conséquences d'une explosion. Nous avons vérifié à chaque fois que l'explosion parcoure une case si elle rencontre une brique, un item ou une autre bombe.

Rencontre d'une brique ou d'un item : on indique que l'entité doit se casser à la fin de l'explosion (grâce à un booléen), et on récupère la position de cette entité. Ainsi, on aura accès à cette entité plus tard afin de la faire disparaître de la carte.

Rencontre d'une autre bombe : on recherche l'élément qui représente cette bombe dans la liste des bombes posées (grâce à ses positions), puis on effectue deux actions. D'abord on prend cet élément et on le place en premier dans sa liste. Ainsi, ce sera cet élément dont le délai sera vérifié à ce moment-là. Enfin, on modifie manuellement l'instant auquel il a été posé de manière à ce que son délai soit dépassé et que le programme fasse exploser cette bombe directement. L'illusion est immédiate : on a l'impression que l'explosion de la première bombe a entraîné l'explosion de la seconde.

On peut désormais inscrire dans la carte des flammes selon la portée de chaque direction. Cet affichage doit durer un certain temps (défini en millisecondes dans une macro-constante).
On peut utiliser le même système que précédemment : lorsqu'au moins un élément est présent dans la liste de bombes explosées, alors on vérifie si le délai du premier élément est passé. Si oui, alors deux actions sont effectuées :
- Effacement des flammes de l'explosion grâce à la portée de la bombe.

- Suppression des briques ou des items touchés par les flammes de chaque direction.

ENSEMBLE DES SPRITES UTILISES POUR LES BOMBES :

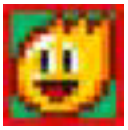

Bonus

Les items bonus sont indispensables dans le jeu. Ils permettent de créer de l'aléatoire, étant donné que chaque brique est susceptible d'abriter un bonus. Ils servent à rendre les personnages plus puissants afin de mettre en place une stratégie de combat. On y trouve différents types de bonus :

- La flamme : agrandit la portée d'une bombe
- La bombe : augmente le nombre de bombes qui peuvent être posées simultanément
- Les roller : augmente la vitesse
- Le pied : donne le pouvoir de pousser des bombes

Une liste chaînée a été nécessaire pour stocker nos items, afin de pouvoir allouer une taille dépendant du nombre d'items à disposer. On pourra également supprimer des éléments de la liste au cours de la partie lorsque le joueur récupère ces items. L'idée a été de répartir N items sur la carte au début de chaque partie. Pour cela, nous avons décidé le nombre d'items de chaque type à disposer sur la carte (définis dans des macro-constantes afin de pouvoir les modifier facilement). On leur attribue à chacun une position sur la carte, en vérifiant que cette position coïncide avec une brique et qu'elle ne coïncide pas avec la position d'un autre item. On recommence jusqu'à obtenir une position pour chaque item bonus à disposer. En même temps, on inscrit dans la liste chaînée la position et le type de chaque bonus.

ENSEMBLE DES SPRITES UTILISES POUR LES ITEMS :


BILAN DES OBJECTIFS

Objectifs atteints

L'objectif principal de notre projet est atteint. En effet, nous avions comme but de réaliser le jeu vidéo Super Bomberman. Et selon nous, cet objectif est atteint. Nous avons réussi à recréer un jeu où, dans une carte, nous pouvons faire déplacer jusqu'à 4 joueurs et les faire s'affronter. Chacun de ces joueurs peut poser des bombes sur son chemin pour détruire des briques et éliminer ses adversaires et même récupérer des bonus.

Tout ce travail est pour nous une grande satisfaction car, au début de ce projet, nous n'étions pas sûr d'arriver à un tel résultat à temps. Bien que tous nos objectifs n'ont pas été atteints, nous avons tout de même validé les plus importants.

Objectifs non atteints

L'accomplissement final de notre projet était de développer une intelligence artificielle. En nous inspirant du mode aventure du jeu Super Bomberman original, nous avons imaginé cette IA sous forme de petits monstres qui se déplacent aléatoirement sur la carte. Le joueur aurait pour but d'éliminer ces petits monstres pour gagner le niveau.

Nous avions pensé à ajouter du son à l'aide de la bibliothèque Fmod. Pour cela, nous aurions dû consacrer du temps à l'apprentissage de cette bibliothèque afin d'inclure tous les effets sonores du jeu.

L'ajout de la fonctionnalité de l'item bonus "Pied" aurait permis aux joueurs de faire glisser des bombes à travers le terrain en les poussant. A notre grand regret, la fonctionnalité n'a pas pu être implémentée. Elle aurait pourtant ajouté un aspect très fun à la partie.

Malheureusement, par manque de temps nous n'avons pas pu programmer les fonctionnalités citées plus haut. En effet, lors de la conception du jeu, nous avons toujours privilégié l'aspect multijoueur aux autres fonctionnalités car c'est le mode qui donne au jeu tout son intérêt.

CONCLUSION

Nous arrivons donc au terme de ce projet. Après plusieurs mois passés à nous creuser la tête, à écrire notre code et corriger nos erreurs, il est temps de rendre notre résultat final. Ce projet a été très enrichissant pour nous deux car il nous a été indispensable d'utiliser toutes les connaissances acquises en C durant les deux dernières années ainsi que d'autres connaissances qui n'ont pas été vu en cours. C'est là que le projet a été très intéressant, car nous nous sommes aventuré dans un domaine qui n'a pas été enseigné par le professeur, à savoir la SDL, ce qui fait que nous étions plus autonomes. Nous ne pouvions être aidés que par nous-même.

Certes, tous nos objectifs n'ont pas été atteints, mais nous ne sommes pas déçus car d'une part, notre résultat est très satisfaisant à nos yeux. D'autre part, le projet s'achève, mais nous n'avons pas l'intention pour autant de nous arrêter là. Nous avons bien l'intention, si le temps nous le permet, arriver au bout de nos objectifs.

LISTING

Ce qui fonctionne

- Une carte similaire à l'originale
- Gestion des collisions
- Bombes explosant 3 secondes après avoir été posées
- Mode multijoueur
- Item pour booster le personnage (roller, flammes, bombes multiples)
- Personnage meurt après avoir été touché par une bombe
- Les bombes cassent les murs qu'ils touchent et activent les autres bombes qu'elles touchent
- Menu qui permet de choisir le mode de jeu

Ce qui ne fonctionne pas

- Mode 1 joueur (pas d'IA)
- Problème avec le joueur 4 qui ne peut pas poser de bombe après avoir éliminé quelqu'un
- Pas de son
- Manque du bonus « Pied » pour pousser les bombes

ANNEXE

Openclassrooms.com

Google Images

Developpez.com

Wikipédia

Cours de M. Calcado