**RAPPORT PROJET CASSE-BRIQUE - ARCHITECTURE 2009-2010**

**Introduction**

Ce rapport à pour but de présenter nos travaux, et de guider les intervenants qui nous noterons, dans les différentes tâches que nous avons accomplies afin d'obtenir le produit final, un jeu de casse-brique réalisé en Assembleur à l'aide de l'outil EMU8086.

Dans un premier temps nous présenterons le projet, les membres qui y ont participé et la répartition du travail.

Puis nous parlerons du déroulement du projet. C'est à dire, l'élaboration des différents algorithmes (qui seront expliqués en détails), puis le codage du jeu en assembleur (sous forme de différents jalons). En outre nous expliquerons les différentes difficultés rencontrées. Pour terminer, nous verrons les optimisations et ajouts apportés, ainsi que les ajouts possibles.

Enfin, nous terminerons par une conclusion, expliquant ce que nous a apporté ce projet.

**Partie I - Présentation du projet**

**A. Avant propos**

Ce projet s'inscrit dans la dynamique des projets d'architecture de la promotion B1 de SUPINFO. Le but est de créer un jeu de Casse-Brique en Assembleur.

L'équipe de ce projet est constituée de Clément STROPPA (92692), Gautier LABARRE (93762), Cyril MAITRE (92807), Raphael WALTER (92582) et d'Alexandre DUBOIS (95222).

**B. Répartition du travail**

Le projet s'est déroulé en 4 étapes.

**1. Elaboration des algorithmes.**

Lors de cette étape, nous avons réfléchi à la conception du casse-brique, c'est à dire à l'élaboration des principaux algorithmes.

Nous avons tous participé à cette partie.

**2. Ecriture du code en Assembleur.**

Ensuite nous avons commencé à coder le jeu (en Assembleur), à l'aide des algorithmes précédemment écrit, et en découpant le jeu en jalon (comme indiqué sur le cahier des charges).

Cette étape a été réalisée par Cyril et Raphael.

**3. Ajout et Optimisation**

Pendant cette troisième étape, nous avons ajouté et optimisé quelques aspects du jeu, pour fournir un produit de meilleure qualité. Par exemple nous avons optimisé le déplacement de balle, en ne faisant pas les tests de collision entre les lignes 4 et 20 (car il n'y a aucune brique entre ces lignes)

Nous avons tous participé à cette partie.

**4. Finalisation du projet**

Pour finir, nous avons écrit les différents documents de rendu. C'est à dire ce rapport et le manuel d'utilisation. Mais aussi le PowerPoint pour la soutenance, ainsi que divers fichiers expliquant nos choix théoriques (vous trouverez ces fichiers dans les différents dossiers de jalons).

Nous avons tous participé à cette partie.

**Partie II - Elaboration des algorithmes:**

Avant de commencer le code, nous avons pris le temps de réfléchir et élaborer des algorithmes.

**1. Déplacement de la balle**

Tout d'abord nous nous sommes intéressés au déplacement de la balle. Pour ceci nous utilisons le registre DI qui contient la position (la case) de celle-ci à l'écran. Il faut comprendre que l'écran est constitué de 25 lignes et de 80 colonnes, ceci peut donc être vu comme une matrice 25 sur 80, soit 2000 cases. Ainsi en utilisant la mémoire graphique (en affectant la valeur 0b800h à DS), on peut accéder à chaque case de l'écran avec l'instruction ds:[xx] (où xx représente le numéro de la case. De plus "ds:" peut être omis car c'est la valeur par défaut).

Il faut aussi comprendre 2 choses. La première est que l'on accède à une case avec un seul indice (et non une ligne + une colonne). Autrement dit, pour accéder à la colonne 0 de la ligne 1, il faudra en fait accéder à la case 80 (les cases 0 à 79 représentant la ligne 0).

De plus, une case est définie sur 2 octets, le premier contient le code ASCII du caractère à afficher, le second contient un code couleur qui représente la couleur de fond et du caractère de la case. Ainsi [0] permet d'indiquer le caractère à afficher dans la première case (colonne 0, ligne 0), et [1] permet d'indiquer les couleurs de cette première case. Le caractère de la seconde case (ligne 0, colonne 1), est stocké dans [2], et sa couleur dans [3]. Et de même pour les cases suivantes.

En conclusion, pour afficher un caractère à la ligne x et colonne y, il faudra accéder à [x\*160 + y\*2] (sachant que l'on commence bien sur à la ligne 0 et colonne 0). Donc si la balle descend d'une ligne et bouge d'une colonne vers la droite, il faudra ajouter à DI (qui représente sa position) 160 (pour ajouter une ligne) et 2 (pour ajouter une colonne). Ainsi si la balle se trouvait à la ligne 2 et colonne 5 (donc DI vaut 2\*160+5\*2 = 330), DI vaudrait alors 330 + 160 + 2 = 492, soit la ligne 3 colonne 6.

**2. Rebond de la balle**

Une fois l'algorithme de déplacement de la balle établi, nous avons réfléchi aux rebonds de celle-ci.

Dans un premier temps sur le cadre du jeu. Pour cela, à chaque fois que la balle se déplace on incrémente / décrémente sa ligne et colonne courante. Par exemple si la balle se déplace en bas à gauche, à chaque déplacement on incrémente la ligne courante de la balle de 1 et on décrémente la colonne courante de la balle de 1. Ensuite on test si la ligne courante est égal à 0, si oui la direction verticale de la balle est alors BAS. Mais aussi si la ligne courante est égale à 24 (dernière ligne), si oui la direction verticale de la balle est alors HAUT. De même pour les colonnes, on teste si colonne courante est égal à 0 ou 79 et on change la direction horizontal en fonction des tests. De plus, lors de l'implantation du cadre il suffit juste de changer ligne minimum et maximum (0 -> 23), et colonne minimum et maximum (0 -> 77).

Dans un deuxième temps, nous avons réfléchi au rebond sur les briques. Pour cela nous avons utilisé la technique suivante:

A chaque déplacement de balle, 8 tests sont réalisés. Tout d'abord pour savoir si une brique se trouve au dessus ou en dessous de la brique. Pour cela, il suffit de vérifier que la case au-dessus ou en-dessous contienne le caractère ASCII 219 (qui représente une brique). Si oui, la direction verticale de la balle est changée. De même pour savoir si une brique est à droite ou à gauche de la balle, la direction horizontale (droite ou gauche) sera modifiée. Ensuite, 4 tests permettent de voir si une brique se trouve en haut à droite, en haut à gauche, en bas à droite ou en bas à gauche. Comme avant il faut vérifier si les cases correspondantes contiennent le caractère ASCII 219, mais il faut aussi analyser la direction de la balle. En effet, si une brique se trouve en haut a droite de la balle, mais que cette dernière se dirige en bas à droite, il n'y a aucune raison qu'elle rebondisse dessus. Le même principe est appliqué à chaque cas.

**3. Déplacement de la raquette**

En outre, nous avons un algorithme pour le déplacement de la raquette. Ceci est relativement simple puisqu'à chaque déplacement de balle, il suffit de vérifier si une touche a été pressée. Si c'est le cas, on l'analyse. Si la touche pressée est la touche "**w**", il faut vérifier que la raquette n'est pas complètement à gauche (si c'est le cas, elle ne doit pas se déplacer) en comparant le numéro de la case à l'extrémité gauche de la raquette avec la case à l'extrémité gauche de l'écran de jeu (à savoir 3842 / 2 = 1921 cases). Si le test est négatif il faut alors effacer l'extrémité droite de la raquette et rajouter un caractère à gauche de la raquette (de cette façon on ne droit pas l'effacer puis la réafficher). Il en va de même lorsque l'on appuie sur la touche "**x**".

**4. Affichage des statistiques**

Concernant les algorithmes pour l'affichage du score, du nombre de vies, du message "gagné" et "perdu", nous avons aussi utilisé la mémoire graphique. Comme expliqué avant, il suffit alors d'accéder aux cases adéquates et d'y mettre le code ASCII de la lettre (ou chiffre) que l'on veut afficher.

**5. Affichage des nombres**

Pour finir, voyons l'algorithme pour afficher les nombres. Pour les valeurs de 0 à 9 (un chiffre), il suffit d'ajouter à la valeur du chiffre, 48 (qui correspond au code ASCII du chiffre 0). Par exemple pour afficher 3, il faut en fait mettre dans la case voulue le code ASCII 51 (3 + 48) qui correspond bien au chiffre 3 dans la table ASCII. Pour les nombres à plus d'un chiffre, il suffit d'utiliser l'instruction DIV. En effet cette instruction permet d'obtenir le quotient et le reste. Ceci permet d'obtenir le nombre d'unités, de dizaines, de centaines, etc d'un nombre.

Par exemple, pour afficher 174 :

On divise 174 par 100, on obtient un quotient de 1 et un reste de 74. On affiche le chiffre 1 tout à gauche (avec la technique citée précédemment), puis on divise 74 par 10. On obtient un quotient de 7 et un reste de 4. Ainsi on affiche le chiffre 7 à droite du chiffre 1 et enfin le chiffre 4 tout à droite. En conclusion, il suffit de diviser un nombre par des puissances de 10 dégressives, jusqu'à obtenir un reste inférieur à 9. Puis il suffit d'afficher tout les quotients et le reste final dans l'ordre (les quotients seront tous inférieurs à 9 car on commence avec la puissance de 10 égale à celle du nombre, par exemple pour 1859, on commencera par diviser par 1000, 103).

*Note: Vous trouverez tous ces algorithmes, ainsi que certains détails techniques supplémentaires dans le commentaire du code.*

**Partie III - Codage du jeu en jalon**

**A. Code - Les différents Jalons**

Une fois la phase d'écriture des algorithmes terminée, nous avons commencé à coder le jeu sous forme de jalons.

**1. Jalon 1**

Le code de ce jalon est dans le dossier "Jalon 1". Il permet d'afficher un trait oblique qui représente le chemin de la balle.

**2. Jalon 2**

Le code de ce jalon est dans le dossier "Jalon 2". Il permet d'afficher le chemin de la balle lors de ses rebonds. La balle rebondit à l'infini (ici sont mis en place les tests du cadre).

**3. Jalon 3**

Le code de ce jalon est dans le dossier "Jalon 3". Il permet d'afficher le déplacement de la balle (sans le tracé, qui sera effacé). Donc dans ce jalon la balle se déplace sur tous l'écran en rebondissant automatiquement sur les rebords de la fenêtre.

**4. Jalon 4**

Le code de ce jalon est dans le dossier "Jalon 4". Il permet d'afficher une croix au centre de l'écran. Cependant la balle ne rebondit pas encore dessus.

**5. Jalon 5**

Le code de ce jalon est dans le dossier "Jalon 5". Ce code permet à la balle de rebondir sur la croix mais aussi d'effacer les briques adjacentes à la balle.

**6. Jalon 6**

Le code de ce jalon est dans le dossier "Jalon 6". Dans ce jalon nous avons ajouté le cadre du jeu, ainsi que le score, niveau et brique restante. En outre, nous avons ajouté la raquette, et un système de niveau. Les collisions haut à droite - gauche ou en bas à droite - gauche, sont maintenant gérées.

*NB : La croix, a été enlevée de ce jalon.*

**B. Les difficultés**

Durant ce projet, nous avons rencontré plusieurs difficultés.

**1. L'interface graphique**

L'effacement des éléments graphiques était assez difficile à optimiser.

**2. Les collisions**

Les collisions entre la balle et les briques ont aussi posé quelques difficultés. En effet, la gestion des collisions avec des briques situées à droite ou gauche, et haut ou bas était relativement simple. Cependant, pour les briques situées en haut a droite - gauche et en bas à droite - gauche, de la balle, cela a demandé plus de réflexion.

**3. Les événements**

La gestion des événements a aussi été compliquée à gérer. En effet, il fallait utiliser les bonnes interruptions, et implémenter les bonnes analyses.

**C. Optimisation**

Une fois le jeu terminé, nous avons ajouté quelques optimisations.

Nous avons optimisé le déplacement de la balle en ne faisant pas les tests de collisions entre les lignes 4 et 20 (il n'y a aucune brique, et la raquette se trouve sur la ligne 21).

Nous avons aussi optimisé l'affichage d'éléments graphiques, en utilisant des boucles au lieu d'afficher les éléments un par un.

Enfin nous avons optimisé les tests en utilisant des flags. Ces flags changent en fonction des résultats des différents tests, ce qui influe le jeu (par exemple, le déplacement de la balle).

**D. Ajout**

Nous avons aussi ajouté quelques éléments pour améliorer le jeu :

**Le menu principal**, qui permet de commencer une nouvelle partie, ou bien de quitter le jeu.

**Le nombre de briques restantes**, qui indique le nombre de brique de doit détruire le joueur pour gagner le niveau. Ceci permet d'éviter au joueur de détruire toutes les briques de l'écran (ce qui serait très long).

**La croix**, qui a été supprimée pour des raisons esthétiques. Cependant elle est présente dans les jalons 4 et 5. De plus le code pour l'affichage de la croix est encore disponible dans le jalon 6 (init\_croix).

**Partie IV - Conclusion:**

**A. Objectifs atteints ?**

Oui, les objectifs ont été atteints. Le jeu respecte le cahier des charges. De plus nous avons ajouté un menu principal.

Le jeu fonctionne sous Emu8086 1.64, sous différents PC, ce qui nous permet de dire que le jeu est stable et fonctionnel.

Concernant le répertoire à rendre, celui-ci contient tous les objets demandés.

**B. Coût de la solution**

La durée de réalisation du projet est d'environ 100 heures. De plus, en tenant compte du fait qu'il s'agit de notre premier projet, nous pouvons déterminer un prix à l'heure d’environ 10 €. Soit un prix égal à 1000 € pour la solution complète.

**C. Ce que nous a apporté ce projet**

Ce projet nous a apporté de l'expérience sur de nombreux points. Il nous a appris à travailler ensemble, à se partager les différentes tâches et ensuite à mettre en commun le travail de chacun.

Ensuite, il nous a permis d'approfondir le langage Assembleur et l'utilisation de Emu8086 1.64., mais aussi d'apprendre le fonctionnement d'un ordinateur (fonctionnement de la RAM et du processeur).

Enfin, nous avons appris à prendre de bonnes décisions qui ont parfois été capitales dans l'accomplissement à bien de ce projet, mais aussi à être soumis à des contraintes de temps et techniques.

En conclusion ce projet à été riche en apprentissage et nous a procuré une expérience non négligeable pour notre futur métier, Ingénieur en Informatique.