Livrable n°1

Groupe n°45 – Bubble Blast II

EL BAGDOURI Imane

MOUSEL Léa

CASEZ Baptiste

TOPIN Edouard

Résumé

Ce pré-rapport a pour but de vous amener à découvrir notre projet informatique : le développement en langage C d'une version du célèbre jeu Bubble Blast II. Ce projet aura une double vocation. D'une part, il apparait comme évident que l'aspect technique nous permettra d'enrichir nos connaissances en programmation et d'aborder des une large palette des caractéristiques de la conception d'une application (codage, interface graphique, etc… Par ailleurs, un autre point important de ce projet sera la gestion de celui-ci. En effet, un tel projet demande une forte implication de la part des membres de l'équipe et justifie ainsi une bonne coordination du groupe afin d'accroître notre efficacité. Nous allons à présent exposer différents prérequis nécessaires au bon déroulement et à l'aboutissement de ce projet.

Table des matières

Introduction et règles du jeu

Notre objectif est de développer une version d'un célèbre jeu pour smartphone intitulé Bubble Blast II, dont les règles sont expliquées ci-après. Nous avons choisi ce jeu car sa conception est relativement abordable compte tenu de nos niveaux en programmation. Toutefois, ce jeu nous offre la possibilité de toucher à une large palette de domaines de programmation : algorithmes de déplacements des bulles, concepts graphiques,…

Le but de ce jeu ludique est de faire disparaître toutes les bulles présentes à l'écran.
Pour cela, le joueur déclenche une réaction en chaîne. En effet, chaque bulle possède un rang qui lui est propre et correspondant au nombre de clics nécessaires avant qu'elle explose.
Lorsqu'une bulle explose, elle génère quatre billes allant dans les quatre directions possibles (comme illustré ci-après). Ensuite, lorsqu'une bille rencontre une nouvelle bulle, elle décrémente le rang de celle-ci, entraînant ainsi la réaction en chaîne.
Le but ultime est alors de terminer tous les niveaux en minimisant à chaque fois le nombre de coups nécessaires pour terminer le niveau.

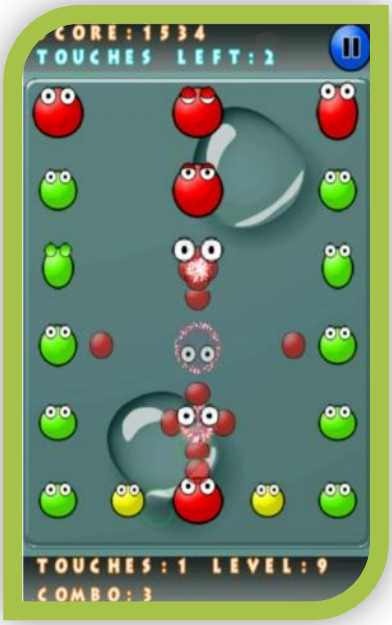


Illustration d'un tableau de jeu

Cahier des charges

Le but ultime est alors de terminer tous les niveaux en minimisant à chaque fois le nombre de coups nécessaires pour terminer le niveau.

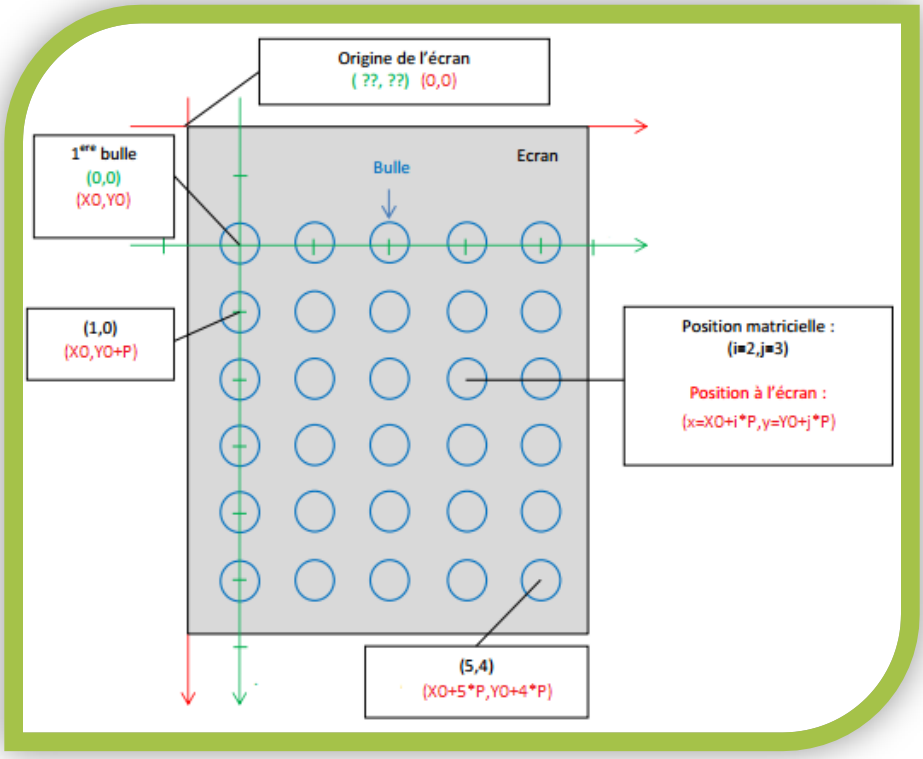
Nous allons à présent développer divers points concernant la mise en place de la programmation de notre jeu Bubble Blast.
Notre objectif étant de développer un jeu complet et offrant une interface jeu-joueur assez riche, notre cahier des charges est plutôt conséquent, en voici les détails.

Tout d'abord, nous souhaitons que le jeu respecte les règles du jeu original Bubble Blast II tout en offrant une certaine facilité de prise en main et d'utilisation du jeu.
Par ailleurs, nous souhaitons développer un jeu interactif, offrant différentes options. Ainsi, nous voulons que l'utilisateur ait la possibilité de choisir la difficulté du niveau, de pouvoir recommencer ou interrompre la partie à tout instant, de pouvoir gérer les différents menus ainsi que de pouvoir couper ou non la musique du jeu.
Pour une bonne clarté et une ergonomie accrue, nous souhaitons adopter un système de représentation explicite des bulles, permettant au joueur de déterminer aisément le nombre de coups restants avant l'explosion de celles-ci.
Nous souhaitons aussi développer un système de sauvegarde permettant d'une part de garder en mémoire les niveaux débloqués par l'utilisateur et d'autre part, de pouvoir reprendre une partie ultérieurement.
Par ailleurs, le jeu étant initialement prévu pour smartphones tactiles, nous avons pris la décision d'offrir pour seuls outils de commande la souris et le bouton « Echap » du clavier.
De plus, la dynamique visuelle du jeu est un point important. En effet, nous souhaitons que visuellement parlant, les mouvements des billes et des bulles paraissent continus.
Dans la mesure où certains niveaux se révèlent difficiles à résoudre, nous souhaitons mettre en place un système de solveur que l'utilisateur peut requérir afin de se débloquer lorsqu'il n'arrive pas à terminer par lui-même.
Enfin, nous avons choisi de fixer deux constantes afin de dimensionner la taille des fenêtres de jeu qui sera donc fixe. Le joueur ne pourra donc pas réduire ou agrandir la taille des fenêtres, ce qui n'est pas nécessairement un frein concernant l'ergonomie de notre jeu.

Vocabulaire

Par souci de clarté, nous vous proposons une brève synthèse du vocabulaire que nous utiliserons dans la suite de ce rapport.

Sur l'écran du joueur apparaissent des ronds statiques de couleurs et de tailles différentes. Le terme bulle désignera ce type de rond. Lorsqu'une bulle explose, elle se décompose en 4 petits éclats que nous nommerons billes.
Nous désignerons par tableau de jeu la fenêtre l'ensemble des bulles, billes et nombre de clics restants. Plus généralement le tableau de jeu pourra désigner la fenêtre où sont affichés bulles, billes, nombre de clics, niveau, etc.
Le rang d'une bulle désignera le nombre de coups à lui appliquer avant qu'elle explose. Le rang zéro indiquera que la bille n'existe plus.
Une page correspondra à ce qui s'affiche dans la fenêtre du joueur : menu, options, choix du niveau, tableau de jeu, etc. Ces différentes pages permettront de faciliter le dialogue avec l'utilisateur.
A chaque partie les bulles sont réparties sur une grille de longueur 6 bulles et de largeur 5 bulles. L'ensemble pourra donc être vu comme une matrice de taille 6x5 dont les coefficients correspondront aux rangs des bulles du tableau de jeu. Nous appellerons cette matrice la matrice bulles. Ainsi à un coefficient (i,j) de la matrice bulles est associé une position (x,y) à l'écran à savoir celle de la bulle dont le rang est égal au coefficient (i,j) de la matrice bulle. Par souci de clarté, la position à l'écran désignera la position SDL (x,y) tandis que la position matricielle désignera la position (i,j) dans la matrice bulles. Enfin nous utiliserons l'appellation position courante pour indiquer la position de la bille que nous considérons.



Le schéma précédent résume le vocabulaire que nous

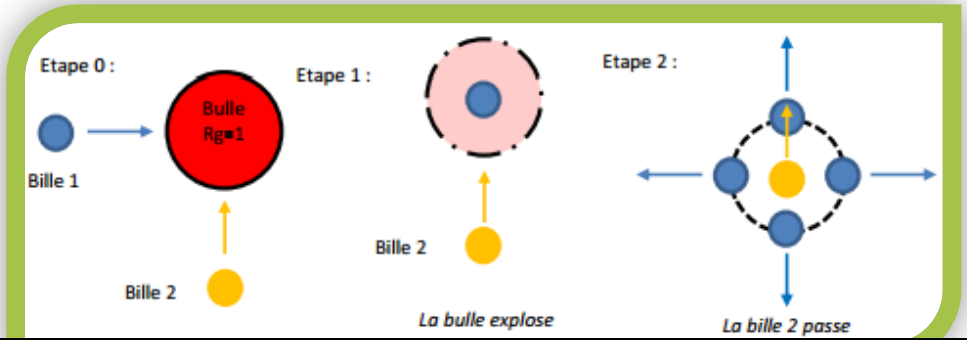
venons de vous présenter. Le repère associé aux positions à l'écran (en rouge) et celui aux positions matricielles (en vert).

Remarque: P correspond au nombre de pixels séparant deux bulles.
Il est à noter enfin qu'à toute position matricielle peut être associée une position à l'écran. La réciproque est fausse puisque ces positions sont dans N^2.

Préparation de la programmation

Avant de se lancer dans la programmation, il est nécessaire de poser le problème et d'en faire une analyse précise.
Il y aura 3 grands types d'événements à gérer : l'affichage, la gestion du jeu en lui-même (déplacement des billes et réactions en chaîne) et l'écoute des demandes de l'utilisateur.
Demande de l'utilisateur : Les demandes de l'utilisateur doivent pouvoir être prises en compte à tout moment. En particulier le déplacement des billes ne doit pas bloquer tout le programme. En effet, le langage C est un langage linéaire dans le sens où plusieurs événements ne peuvent être gérés simultanément. Ainsi le déplacement des billes se fera par déplacements élémentaires, dont le pas doit être suffisamment petit pour que le mouvement semble continu aux yeux du joueur.
Disparition de billes : on ne peut pas anticiper, lors de la création d'une bille, où elle va disparaître. Il sera donc nécessaire de suivre son évolution au cours du temps.

Cas particulier : les billes seront déplacées les unes après les autres, ce qui peut être problématique dans la situation suivante :



Etape 0 : Lors d'un déplacement élémentaire, 2 billes arrivent simultanément sur une bulle de rang 1
Etape 1 : La bille 1 fait exploser la bulle (car elle est de rang 1) : la bille 1 disparaît donc tandis que 4 nouvelles billes sont créées
Etape 2 : La bille 2 est déplacée et ne sachant pas qu'il y avait une bulle auparavant elle continue son chemin. Il en résulte donc 5 billes.

Cet exemple met en exergue l'incompatibilité entre le caractère séquentiel de l'algorithme et l'arrivée simultanée de plusieurs billes sur une même bulle.

Affichage : l'écran devra être rafraîchi suffisamment régulièrement pour qu'après chaque nouvel évènement les nouveautés soient affichées. Comme SDL

superpose les images, tout l'écran devra être réaffiché après un déplacement élémentaire des billes afin que les billes avant déplacement ne soient pas affichées.

Avant de concevoir les algorithmes, il est important d'associer aux différentes données, telles que les bulles et les billes, des représentations simples et appropriées.

Gestion des positions : on parlera beaucoup de position et donc de doublets d'entiers. Pour simplifier l'utilisation l'utilisation de ces doublets nous créerons un type POSITION :

```
typedef struct {
        int x, y;
} POSITION ;
```

Ainsi positions matricielles, positions à l'écran et positions courantes pourront être représentées dans ce type.

Les bulles : les bulles seront caractérisées par leur rang et leur position matricielle. Il sera donc intéressant par la suite de créer un type BULLE qui regroupera la position matricielle et le rang dans une même structure :

```
typedef struct {
        int rg ;
        POSITION pos ;
        } BULLE;
```

Les billes : elles ne seront pas fixes (par définition) et se déplaceront à vitesse identique. Ainsi chaque bille se caractérisera entièrement par une position à l'écran, la position courante (cf. vocabulaire) et une direction de déplacement. Il sera également intéressant de les caractériser par leur position matricielle future afin de faciliter le cas de disparition des billes. Cette position sera déductible de la position courante et de la direction. Elle pourra être en dehors de la matrice bulles et dans ce cas la bille est vouée à sortir de l'écran. La position matricielle future sera donc un moyen de vérifier si la bille a atteint une bulle. Nous serons donc probablement amenés à créer un type BILLE qui regroupera la position à l'écran, la direction et la position matricielle future dans une même structure :

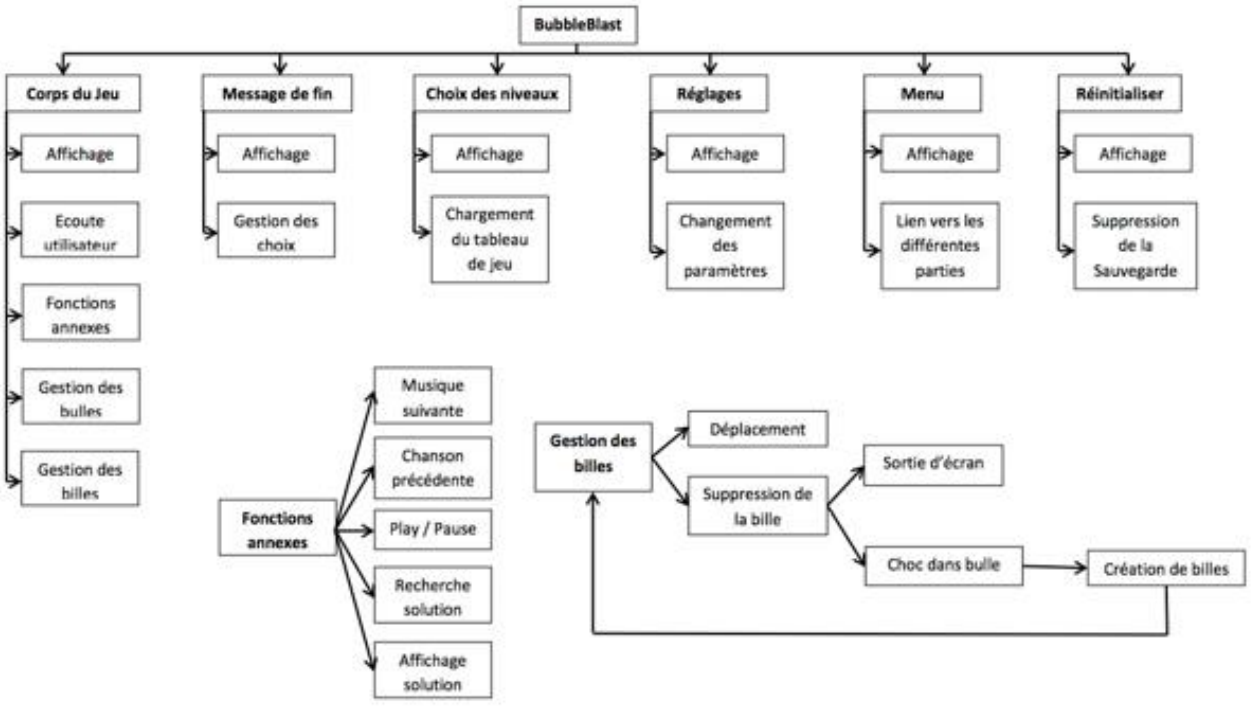
```
typedef struct {
        POSITION pos,dir,futur ;
} BILLE ;
```

Il est à noter que les billes sont en nombre variable. Nous avons commencé à réfléchir à la méthode que nous pourrions employer afin de les stocker. Deux idées nous sont venues pour le moment : stocker les billes dans une pile ou bien les stocker dans un tableau de taille suffisante pour accueillir le maximum de billes (une majoration empirique à 100 nous paraît envisageable). Nous choisirons entre ces deux méthodes celle qui nous paraîtra la plus simple à gérer en fonction du reste de notre algorithme.

Tableau de jeu : Nous envisageons de créer un type TABLEAUJEU rassemblant l'ensemble des bulles, les billes et le nombre de clics autorisés afin de fournir, grâce à une seule variable, toutes les informations nécessaires au bon fonctionnement des fonctions que nous serons amenés à utiliser lors de la programmation.

Analyse fonctionnelle du problème

L'intégralité du jeu va reposer sur différents algorithmes que nous implémenteront en langage C. Nous avons déjà réfléchi à certains d'entre eux.
Un point épineux de la conception, résidera dans le traitement des mouvements des billes et sur leurs arrivées sur certaines bulles, impliquant l'altération du rang de celles-ci En effet, si une bille rencontre une bulle possédant un rang supérieur à 2, il suffira de décrémenter le rang de la bulle et de faire disparaitre la bille. Dans le cas où la bille rencontrerait une bulle de rang 1, la tâche se complique puisqu'il faut faire disparaître la bulle et la bille et générer les 4 nouvelles billes relatives à l'explosion de la bulle.
En ce qui concerne le solveur, nous pensons pour l'instant à effectuer un algorithme récursif, qui sera certainement coûteux en temps de calcul mais qui pourra évaluer l'intégralité des combinaisons possibles. Ainsi dès qu'une combinaison possible permettra de résoudre le tableau, elle sera proposée à l'utilisateur.
Enfin, pour la boucle principale de notre programme, Nous pensons qu'il serait judicieux de subdiviser celle-ci en fonction de la commande demandée par le joueur (jeu, gestion du menu, réglages, etc.). Ainsi, le passage d'une fonctionnalité à une autre sera plus aisé et le fonctionnement de ces fonctionnalités sera indépendant par rapport à celui des autres.
Nous allons désormais vous présenter l'analyse fonctionnelle, à proprement dit.



Commençons tout d'abord par l'analyse du corps du jeu.

Affichage :

But : rafraîchissement de l'écran après chaque clic du joueur

Moyen : la fonction afficherTableauJeu

Input(s) : clic du joueur sur une bulle ou sur le bouton faisant appel au solveur

Output(s) : modification du contenu de la structure affichage

Limites : L'écran doit se rafraîchir suffisamment régulièrement pour qu'après chaque déplacement élémentaire des billes, les billes avant déplacement ne soient pas affichées.

Ecoute utilisateur :

But : répondre aux choix du joueur durant la partie

Moyen : la gestion des bulles et des fonctions annexes

Input(s) : choix de l'utilisateur durant la partie

Output(s) : changement du rang de la bulle choisie par le joueur, changement de la musique ou affichage de la solution.

Fonctions annexes :

But : offrir au joueur les deux options suivantes : choisir la musique jouée (voir l'éteindre) et avoir recourt au solveur.

Moyen : module des fonctions annexes pour gérer la musique et le solveur

Input(s) : pression de la part du joueur sur un bouton de la partie supérieur à l'écran, afin d'agir sur la musique ou obtenir l'aide du solveur.

Output(s) : musique changée ou solution proposée.

Gestion des billes :

But : créer, déplacer, puis supprimer les billes

Moyen : module de gestion des billes

Input(s) : explosion d'une bulle

Output(s) : les billes sont générées et déplacées dans les quatre directions, puis les billes sont supprimées lorsqu'elles entrent en contact avec une bulle.

Limite : le nombre de billes générées dépend de la position de la bulle qui explose.

Gestion des bulles :

But : diminuer ou faire disparaître la bulle sélectionnée par le joueur en fonction de son rang.

Moyen : la fonction supprimerBulles et la fonction diminuerRg

Input(s) : Sélection d'une bulle par le joueur avec un clic ou arrivée d'une bille à la position de la bulle.

Output(s) : Bulle avec un nouveau rang (égal à 0 si la bulle explose)

Limite : gérer la priorité entre arrivée des billes et gestion des bulles lorsque deux billes arrivent simultanément.

Fin du jeu :

But : renvoyer le joueur vers le message de fin lorsque la partie est terminée

Moyen : fonction finPartie

Input(s) : Le nombre de clics restants est 0 ou toutes les bulles sont de rang inférieur ou égal à 0.

Output(s) : message de fin

Dépendance(s) : Aucune

Passons à présent au module lié à la gestion des billes.

Initialisation du stockage des billes :

But : initialisation de l'ensemble des billes : Le but est d'initialiser la structure conservant l'ensemble des billes au début du jeu.

fonction ensBillesInit

Input(s) : ENSBILLES

Output(s) : ENSBILLES

But : Déplacement : toutes les billes peuvent se déplacer. La vitesse de déplacement est la même pour chacune des billes.

Moyen : fonction deplacerBilles.

Input(s) : ENSBILLES, qui correspond au tableau contenant l'ensemble des billes.

Output(s) : ENSBILLES

Dépendance(s) : deplacerBille, supprimerBille, creerBille, diminuerRangBulle.

Ajout des billes :

But : ajout des billes : L'ajout de 4 billes a lieu lorsqu'une bulle éclate.

Moyen : fonction ajouterBilles.

Input(s) : BILLE.

Output(s) : 4 BILLE[S].

Dépendance(s) : Aucune

Suppression d'une bille :

But : suppression d'une bille : suppression d'une bille dans le cas où elle sortirait de l'écran ou qu'elle croiserait une bulle

Moyen : fonction supprimerBille

Input(s) : BILLE, correspondant à une bille créée précédemment.

Output(s) : BILLE

Dépendance(s) : Aucune

Suppression des billes :

But : suppression des billes après un tour de l'algorithme : le but est de prendre en compte la disparition de toutes les billes devant disparaitre avant l'actualisation de la fenêtre de jeu.

Moyen : fonction supprimerBilles

Input(s) : ENSBILLES, qui correspond au tableau contenant l'ensemble des billes.

Output(s) : ENSBILLES

Dépendance(s) : supprimerBille

Désormais, intéressons nous au détail des fonctions annexes.

Changement de musique

But : Pouvoir changer la musique lors d'une partie.

Moyen : fonctions music_suiv et music_prec, qui permettent de changer le morceau en lecture lors de la partie.

Input(s) : Pression de la part de l'utilisateur sur un bouton graphique présent dans le menu.

Output(s) : Changement du morceau sélectionné pour le jeu.

Limites : Nous allons nous limiter à un nombre restreint de musiques (min 2 - max. 5). La modification aura lieu pendant le jeu et dans le menu. La musique ne sera valable que pour l'instant de jeu, la musique du reste sera fixée afin de mieux distinguer le jeu du menuing

Marche-arrêt de la musique :

But : Pouvoir arrêter/relancer la musique

Moyen : fonctions play_music et stop_music.

Input(s) : Pression de la part de l'utilisateur sur un bouton graphique présent dans le menu et dans le jeu.

Output(s) : Activation ou non de la musique

Limites : Nous souhaitons avoir un accès permanant à cette option. C'est pour cette raison, que cette dernière sera présente dans le menu et dans le jeu. Son action s'étend à la musique du menu. Le jeu à défaut joue une musique.

Recherche de la solution :

But : chercher une solution (si elle existe) à l'instant où l'utilisateur le demande.

Moyen : fonction solver ainsi qu'un bouton afin de chercher une solution.

Input(s) : plateau de jeu en cours, nombre de billes, nombre de coups restants.

Output(s) : Booléen pour savoir si le niveau est résoluble, ainsi qu'un tableau indiquant la solution.

Limites : La fonction va tester toutes les possibilités. C'est pour cela que nous limitons le rg d'une bille. La recherche de solution n'est valable que si aucunes billes ne bougent.

Affichage du coup suivant :

But : Afficher à l'utilisateur le prochain coup à faire.

Moyen : animer un halo autour de la bulle à éclater s'il existe une solution (sinon afficher un message).

Input(s) : Booléen de solver et tableau de solver.

Output(s) : afficher un halo sur la bulle suivante ou déclencher le message de fin de partie.

Limites : se déclenche suite à la fonction solver.

En ce qui concerne la partie graphique, nous avons fait l'analyse suivante :

Message de fin :

But : afficher à l'utilisateur un message de fin lorsque le joueur a fini un niveau afin d'une part de lui indiquer s'il a gagné ou s'il a perdu et d'autre part de lui laisser le choix de poursuivre ou non le jeu. Deux cas distincts sont à distinguer.
Si le joueur a gagné : (messages de fin gagnant)

- Afficher à l'écran : « Bien joué !! », le nombre de clics de souris utilisés par le joueur pour réussir ce niveau, le numéro du niveau qu'il vient de réussir et les trois possibilités qui s'offrent désormais à lui : « réessayer », « niveau suivant » et « quitter ».
- Si le joueur a perdu : (message de fin perdant) Afficher à l'écran, « Dommage ! », le nombre de clics de souris utilisés par le joueur au cours de ce niveau, le numéro du niveau qu'il vient de tenter et les deux possibilités qui s'offrent désormais à lui : « réessayer » et « quitter ».

Moyen : fonction messageFin
Input(s) : booléen de la fonction resultatNiveau, nombre de clics de souris utilisés par le joueur et numéro du niveau.

Output(s) : afficher le message de fin gagnant si booléen de la fonction resutatNiveau est VRAI et afficher le message de fin perdant si booléen de la fonction resultatNiveau est FAUX.
Limites : se déclenche suite à la fonction resultatNiveau.

Gestion des choix :

But : rediriger le joueur selon le choix qu'il effectue.
Moyen : fonctions fermerPage et fonction ouvrirPage.
Input(s) : page en cours et choix du joueur par un clic de souris.
Output(s) : page choisie par le joueur.
Limite(s) : se déclenche suite à la fonction afficherMessageFin.

Choix des niveaux :

But : Afficher au joueur les niveaux qu'il a débloqué et qui lui sont donc accessibles en vert et les niveaux qu'il n'a pas encore débloqués et qui ne lui sont donc pas encore accessibles en gris.
Moyen : fonction afficherNiveaux.
Input(s) : l'ensemble des niveaux (on pourra les mettre dans un tableau, chaque case de ce tableau correspondant à un niveau).
Output(s) : afficher en vert les niveaux débloqués et en gris ceux qui ne sont pas encore débloqués.
Limite(s) : nécessite une sauvegarde des niveaux débloqués et nécessite en aval une fonction debloquerNiveau qui débloque le niveau suivant lorsque ce niveau-ci est réussi par le joueur.

Chargement du tableau de jeu :

But : rediriger l'utilisateur sur la page du tableau de jeu que l'utilisateur a choisi.
Moyen : fonction fermerPage et fonction ouvrirPage.
Input(s) : page en cours et choix du joueur par un clic de souris.
Output(s) : page choisie par le joueur.
Limites : se déclenche à la suite d'un clic de souris de l'utilisateur.

Menu :

But : afficher à l'utilisateur un menu lui proposant de jouer, d'accéder aux réglages ou de quitter le jeu.
Moyen : fonction afficherMenu
Input(s) : clic de l'utilisateur sur un lien le redirigeant sur le menu (« quitter » à la fin d'un niveau ou démarrage du jeu.
Output(s) : affichage de la page du menu.

Lien vers les différentes parties :

But : rediriger l'utilisateur sur la page que l'utilisateur a choisi.

Moyen : fonction fermerPage et fonction ouvrirPage.
Input(s) : page en cours et choix du joueur par un clic de souris.
Output(s) : page choisie par le joueur.
Limite(s) : se déclenche à la suite d'un clic de souris de l'utilisateur.

Réinitialiser :

But : afficher un message à l'utilisateur pour s'assurer qu'il est sûr de vouloir supprimer la sauvegarde de ses données (niveaux débloqués).
Moyen : fonction afficherMessageSuppression .
Input(s) : clic de l'utilisateur sur le bouton graphique « Réinitialiser ».
Output(s) : messages à l'intention de l'utilisateur : « Etes-vous sûr de vouloir supprimer la sauvegarde de vos données ? OUI NON ».
Limites : se déclenche suite à un clic de souris de l'utilisateur.

Suppression de la sauvegarde :

But : suppression de la sauvegarde.

Input(s) : message et clic de la souris sur le bouton graphique « OUI ».
Output(s) : message fermé seul le niveau 1 n'est pas bloqué.
Limites : se déclenche suite un clic de la souris de l'utilisateur.

Annulation de la suppression :

But : fermer le message sans supprimer la sauvegarde
Moyen : fonction fermerMessage.

Input(s) : message et clic de la souris sur le bouton graphique « NON ».
Output(s) : message fermé.
Limites : se déclenche suite à un clic de l'utilisateur.

Réglages :

But : afficher à l'utilisateur les réglages qu'il peut effectuer à savoir activer/désactiver le son et réinitialiser ses données.
Moyen : fonction afficherRéglages.
Input(s) : clic de l'utilisateur sur un lien le bouton graphique du menu « Réglages ».

Output(s) : affichage de la page des réglages.

Changement des paramètres :

But : permettre à l'utilisateur de changer certains paramètres
Moyen : fonction réinitialiser (cf. module ci-dessus) et fonction activerSon
Input(s) : clic sur le bouton graphique « réinitialiser » ou {« activer/désactiver son » + booléen de la fonction sonActif}
Output(s) : message demandant la confirmation dans le cas de la fonction réinitialiser ou dans le cas de la fonction activerSon, si booléen de la fonction sonActif = VRAI alors {bouton graphique « activer/désactiver son » rouge + son désactivé} et si booléen de la fonction sonActif = FAUX alors {bouton graphique « activer/désactiver son » vert + son activé}.
Limites : se déclenche suite à un clic de l'utilisateur et nécessite une fonction sonActif en aval.

Planning de charges prévisionnel

Afin d'aborder les différentes tâches requises à la conception de notre application de manière sereine, nous avons établi un plan de charges prévisionnel que nous comparerons *in fine* à la fiche de suivi d'activité que nous remplirons au fur et à mesure de l'avancement du projet.
Le planning de charges prévisionnel est présenté ci-après.

	Charge en %	Charge en H	El Bagdouri Imane	Mousel Léa	Casez Baptiste	Topin Edouard
Description de l'activité						
TOTAL	100	200	50	50	50	50
Gestion de projet	28.5	57				
Lancement du projet (Réunion avec le tuteur,...)	2	4	1	1	1	1
Planning prévisionnel et suivi d'activité	2.5	5	1	1	2	1
Réunions de suivi	14	28	7	7	7	7
Rédaction (Rapport, comptes-rendus,...)	10	20	4	4	8	4
Spécification	4	8				
Définition du problème et des fonctionnaliés	4	8	2	2	2	2
Conception préliminaire	5	10				
Définition d'un modèle de données	2	4	1	1	1	1
Définition des modules (.h)	3	6	2	1	1	2
Conception détaillée	11	22				
Définition des structures de données	3	6	3	1	1	1
Définition des fonctions et procédures	6	12	3	3	3	3
Définitions des tests unitaires	2	4	1	1	1	1
Codage	23	46				
Ecriture des interfaces (.h)	1	2	0	0	1	1
Ecriture du makefile	1	2	1	1	0	0
Ecriture des fonctions	12.5	25	8	5	5	7
Tests unitaires	8.5	17	4	5	5	3
Interface graphique	13,5	27				
Définition des problèmes	3	6	1,5	1,5	1,5	1,5
Codage via SDL	10,5	21	4	9	4	4
Site Web	9	18				
Auto-formation HTML	4,5	9	2	2	2	3
Codage du site Web	1.5	3	0	0	0	3
Actualisation du site Web	3	6	1,5	1,5	1,5	1,5
Soutenance	6	12				
Préparation de la soutenance	4	8	2	2	2	2
Soutenance	2	4	1	1	1	1