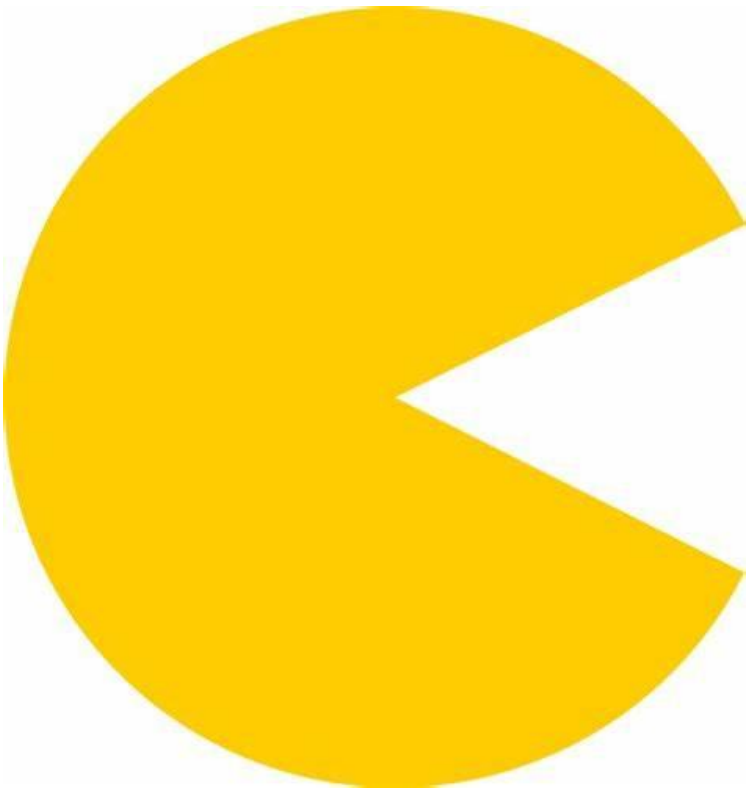
SAE PacMan en C++

Sommaire



Delin Alexis
Lartigaud Elliot
Leroux Mathieu
Oger Léo
Rouge Gabriel

I : Introduction

Après un labeur important et quelques nuits blanches et suite à quelques partiels assez solides, nous sommes fiers de vous présenter l'objet de notre travail à tous les six sur cette SAE. Chacun aura su apporter quelque chose au projet en fonction de ses compétences et ca pacités de travail. Ce rapport servira à prendre une vision générale sur notre devoir, afin de mieux appréhender le programme en lui-même, mais aussi dans l'optique de comprendre de potentielles réflexions importantes qui se cachent derrière ce dépot GitHub et ce tas de fonctions C++, pour tenter de vous offrir une meilleure compréhension de ce que nous vous proposons.

II Organisation générale et préliminaire :

Notre première préoccupation au commencement de ce projet était l'organisation de celui-ci. En effet, grâce à l'expérience accumulée avec les précédents travaux de groupes, l'on a su créer une organisation déjà un peu plus efficace et utile que nos précédentes. Adopter dans les premiers instants une rigueur et une sémantique uniforme a permis une réelle cohésion.

Premièrement, n'a pas été négligé le temps de réflexion sur l'architecture du projet. Vous trouverez en annexe les documents que l'on a produits en préliminaire à de quelconque lignes de codes. Il s'y trouvera par exemple, une liste exhaustive des variables et de leur sémantique que nous avions prévu d'utiliser, ainsi qu'un découpage presque atomique des fonctions afin d'offrir une articulation simplifiée et efficace, et d'autres notes générales sur notre répartition des rôles. Tout cela a permis un début rapide et une participation non-négligeable de chaque élément du groupe.

Ensuite, en cours de production du devoir, des problèmes sont apparus, notamment un souci d'organisation générale du programme qui nous semblait trop lourde à tenir sur un seul fichier. Ainsi, ont été séparé les fonctions sur 8 fichiers différentes. Voici lesquels sont-ce, ainsi que ce qu'ils signifient.

- assertives : Contient la totalité de nos fonctions d'assertions (Les fonctions booléennes qui commencent par is). L'on a en effet fait face à pléthore de fonctions similaires mais subtilement différentes, et les centraliser permettent de mieux les exploiter, et d'éviter d'en reproduire deux identiques.
- gameLogic : Contient tout ce qui touche à la logique basique du jeu ; les déplacements des fantômes et de pacman, les fonctionnalités de manger une bulle ou un fruit, ou de gérer les changements d'état lorsque le joueur attrape une super pac-gum !
- initialization : Ce sont les fonctions qui ne sont appelées qu'une fois, à l'initialisation du programme.
- constants : Ce fichier contient toutes nos constantes, nos plateaux, nos structs, et des éléments graphiques.
- draw : Contient les fonctions concernant l'affichage et l'animation des sprites.
- général : Contient les fonctions susceptibles d'être utiles à n'importe quelle autre partie du programme.
- ghostIntelligence : Contient les fonctions relatives à l'intelligence décisionnelle des fantômes, et seulement à cela.

- param : Permet d'exploiter les données du fichier yaml ; les paramètres.

III Intelligence Décisionnelle

Cette partie concerne l'implémentation de la fonctionnalité de personnalité des fantômes et de l'algorithme A*. Puisqu'il s'agissait de la partie du projet où l'algorithme était le plus complexe, bien que peut être ne demandant peut être pas une grande quantité de connaissances techniques. Ainsi, le temps consacré à l'implémentation de ces fonctions ressemblerait à cela : 60% du débogage, 30% du pseudo-code, 10% de l'implémentation du pseudocode. Par ailleurs, l'entièreté du pseudo code est consultable via l'annexe (notons qu'il ne comporte pas les modifications apportées lors de l'implémentation et du débogage).

Pour ce qui touche aux personnalités, il n'en a été implémenté que trois, bien que l'arborescence du programme permet tout à fait d'en ajouter avec aisance, puisque cela fût pensé ainsi. Les trois personnalités présenes sont : le simplet (dumb), le vénère (hardcore), le confus (confused). L'objectif était d'avoir trois niveaux de difficulté, et selon celui-ci, les fantômes se voyaient avoir des chances plus ou moins grandes d'obtenir une personnalité qui représente un danger ou non. Chaque personnalité a une chance de respecter A*, et cette probabilité augmente avec la difficulté. S'ils ne suivent pas A*, alors ils suivront leur trait de personnalité. Le simplet peut parfois suivre un chemin aléatoire, tandis qu'il arrive au confus de confondre sa cible - pacman - avec un de ses amis fantômes, ou même avec un simple fruit. Enfin, le "vénère", a une chance accrue comparé à ses congénère de suivre A*, soit, il constitue une menace plus importante au joueur. Dans les rares cas où les probabilités font qu'il ne suit pas le comportement A*, alors il prend un chemin aléatoire.

Ensuite, l'algorithme A*. Ce fût une partie du projet des plus longues à implémenter, puisque son pseudo-code et son implémentation uniquement pris environ 8 heures. Il est constitué à lui seul de 11 fonctions ; en notant qu'elles ne se sont pas toutes retrouvées dans le fichier ghostIntelligence. La fonction principale, "aStar", s'occupe de créer toutes les variables nécessaire, et appelle tour à tour les autres fonctions pour former l'algorithme A*. En première partie, est créée une liste de toutes les cases disponibles. Cette liste est prise en paramètre par une autre fonction qui va remplir la variable openNodes. Il s'agit d'une map, qui prend en clé un nœud (donc, une position), et lui accorde en valeur une qualité représentée sous forme de chiffre. Plus basse est la qualité, plus on se rapproche de sa cible ; pacman. Ensuite, l'on fait appel un une fonction qui va remplir closedNodes, il s'agit d'un arbre qui stocke les Position que parcours l'algorithme. En racine est l'emplacement du fantôme, et l'objectif et qu'une feuille atteigne pacman ; ici, la qualité des nœuds permet à l'algorithme de choisir par lequel il passe en priorité. Une fois l'arbre tracé, il faut remonter le chemin de celui-ci depuis la feuille où l'on a trouvé la position de pacman jusqu'à la racine. La fonction renvoie enfin le premier déplacement que le fantôme doit exécuter pour suivre le chemin A*.

Les suites d'instructions peuvent sembler capilotractées à la lecture de cette partie du programme. La cause en est qu'il était nécessaire d'avoir des structs en tant que clé et/ou valeur dans des map. Pour le trop de problème que cela a engendré, il a été décidé de coder ces structs en string afin de pouvoir mieux les stocker ; ils sont décodés de la même manière pour être exploités. Cette manière de procéder entraîne ainsi quelques complexifications, et explique comment certaines lignes de code s'étalent autant.

Malheureusement, A* se révèle ne pas être opérationnel dans toutes les situations, et malgré une quantité importante de tests effectués, nous n'eûmes pas le temps de repérer

les causes de ces dysfonctionnement, ainsi, nous étions dans l'incapacité de déboguer. En conséquence, cet algorithme est présent dans le code source, mais n'est pas utilisé. Cela reste néanmoins une recherche et des réflexions qui furent très instructives, et il n'est pas impossible qu'à l'avenir, ce bout de programme dysfonctionnel se retrouvé exhumé, déboguer, et enfin utilisable et utilisé.

IV : Sprite, audio et transitions

Comme vous avez pu le voir, notre jeu a beaucoup d'éléments uniques. Que ce soit pour les skins, les transitions ou l'audio, tout a été fait par nos soins. Nous voulions vous proposer une expérience moins ennuyante en créant de multiples skins. Dans notre banque de données, nous comportons 32 sprites de fantômes, 64 sprites de PacMan et 8 autres sprites comme les fruits ou les écrans de fin. Au début, nous voulions les créer grâce aux formes présentes dans minGl, mais pour avoir une plus grande liberté d'expression artistique, nous avons pu dessiner tous nos sprites en Pixel art. Un convertisseur en langage Python proposé par vos soins nous a permis de transformer ces fichiers en éléments exploitables par la bibliothèque minGL, et ainsi les implémenter au programme. Bien entendu, ces éléments n'étaient pas une focalisation pour quiconque, puisque le pan le plus important du travail était l'organisation, la conceptualisation, l'implémentation, et le débogage. Mais il s'est avéré que l'implémentation d'autant d'éléments graphiques a soulevé quelques problèmes, et des débats sont nés pour en trouver des solutions ; c'est ainsi qu'on été pensées toutes les fonctions qui animent et exploitent les éléments graphiques fournis.

De plus, nous ne voulions pas utiliser les mêmes musiques de fond que Pacman et avons donc songé à en recréer une de toute pièce, ayant les mêmes sonorités. Celle-ci a été composée depuis le logiciel de création musicale FL Studio. Elle est donc jouée durant toute la partie et modifiée selon les événements qui apparaissent, tels que manger une super pac-gum, gagner ou perdre la partie.

Mais ce point artistique ainsi que les problèmes qui en ont été soulevés ne sont pas le plus important de cette partie. Il s'agit des transitions. Elles ont été mises en place afin que les personnages puissent effectuer des mouvements fluides lorsqu'ils se déplacent de case en case. Ainsi durant toute la partie, après avoir regardé quelle touche était pressée, tous les personnages effectuent un déplacement s'ils en ont la possibilité. Ce déplacement est fluidifié grâce à une transition. Pour éviter de refaire les mêmes transitions en boucles, une variables isTransitionFinished nous indique si la dernière transition qu'à effectué le pacman est terminée. S'il en est le cas, de nouvelles transitions sont relancées pour tous les personnages ; sinon, le jeu attend. Ceci rend le jeu dynamique et plus agréable à jouer, et cette fonctionnalité aura mobilisé une quantité considérable de recherche et d'investissement pour la faire fonctionner.

V Partie Technique

La première priorité de développement n'était néanmoins située sur aucune de ces fonctions. Ils nous était premièrement nécessaire de développer une base solide et exploitable afin de créer quelque chose par-dessus ; cette base solide, c'est la partie

technique, soit, l'arborescence des fichiers, le fichier yaml ainsi que la récupération des paramètres à l'intérieur, le main.cpp avec la suite d'appel de toutes les fonctions nécessaires au fonctionnement du programme et les initialisation de toutes les variables, et enfin, les fonctions centrales comme la récupérations des touches, l'affichage des éléments sur différentes couches pour éviter les incohérences…

Une de nos plus grandes difficultés a été d'avoir eu à gérer des timers. En effet, certains éléments comme l'apparition des fruits ou la durée du "mad mode" nécessite de compter un certain temps avant de pouvoir effectuer une action. Nous avons d'abord essayé de résoudre ce problème en créant plusieurs "threads" ; mais malheureusement, ce fût un échec. Nous nous sommes donc rabattus sur une solution beaucoup moins optimale qui consiste à mettre en place un compteur que l'on incrémente de 1 à chaque passage dans la boucle de jeux principale, ce qui nous permet d'avoir une unité de mesure temporelle. Ainsi, solution était trouvée à ce problème.

Cependant, une explication exhaustive de la partie technique et complexe du projet serait indigeste et péniblement longue, ainsi, cette partie ne se veut que mettre en lumière une partie de nos essais et réflexions.

VI Doxygen

Doxygen est un outil de documentation automatisé pour les programmes informatiques. Il permet de générer des documentations en format HTML, PDF, RTF, XML et man pages à partir de commentaires intégrés dans le code source. Il est compatible avec de nombreux langages de programmation tels que C, C++, C#, Java, Python, PHP, etc.

L'utilisation de Doxygen est simple, il suffit d'ajouter des commentaires spécifiques au format Doxygen dans le code source. Ces commentaires contiennent des informations sur les classes, fonctions, variables et autres éléments du code. Doxygen utilise ces informations pour générer la documentation. Il est également possible de spécifier des options de configuration pour personnaliser la sortie de la documentation.
Les avantages de Doxygen sont nombreux, en voici quelques-uns :

- Il permet de maintenir une documentation à jour automatiquement en suivant les évolutions du code source.
- Il offre une bonne visibilité de la structure du code et de ses fonctionnalités.
- Il facilite la collaboration entre développeurs en fournissant une documentation claire et complète.

En conclusion, Doxygen est un outil pratique et efficace pour la génération de la documentation de code. Il est facile à utiliser et permet de maintenir une documentation à jour automatiquement, tout en améliorant la qualité du code et la collaboration entre développeurs. Il est compatible avec de nombreux langages de programmation et offre une variété d'options de configuration pour personnaliser la sortie de la documentation.

Le sujet de notre propre utilisation et familiarisation avec l'outil Doxygen est aussi une partie pertinente du développement, puisque nos compétences mobilisés dans l'élaboration de celui-ci a porté ses fruits ; la cohésion et l'appropriation du code d'autrui nous a à tous été plus simple dès lors que la doxygen avait été mise en place. Ainsi, nous savons à l'avenir les vertus de l'outil Doxygen dans un contexte de développement et de travail d'équipe, et qu'investir dans celle-ci n'est jamais une perte de temps, spécifiquement lorsque le groupe en question sont familier avec Doxygen ; les économies de temps n'en sont que décuplées.

VII Conclusion

En l'absence de consignes explicites au sujet du rapport, ce document se veut être assez exhaustif pour remplir toutes les attentes que l'on pourrait lui porter. Supposément, après l'avoir lu entièrement, l'esprit devrait être en condition pour comprendre avec un peu plus d'aisance notre programme ; puisqu'aussi bien commenté soit-il, la prise en main du programme d'autrui est toujours désagréable au premier coup d'oeil, et surtout, laborieuse. Le point primaire que l'on a porté à l'organisation, à la sémantique, commentarisation et à la documentarisation via doxygen devrait ainsi compléter les éléments du puzzle pour offrir, peut-être, l'observation d'un programme aussi bien harmonieux que fonctionnel.

L'exécution de ce projet nous aura permis à tous de renforcer nos compétences, évidemment en matière de programmation, mais surtout en matière d'organisation, communication, et travail d'équipe. L'exécution des directives préliminaires à la rédaction du code nous aura permit d'évaluer leur utilité, et de savoir comment mieux les adapter à de futurs travaux, autant que les problèmes auxquels nous avons fait face nous permettent d'en apprendre et de savoir comment les prévenir à l'avenir ; puisqu'on dit bien qu'il vaut mieux prévenir que guérir.

VIII Annexe

Lien du dépot Github :
https://github.com/Badlix/SAE_pacman_Leroux_Rouge_Oger_Delin-A_Lartigaud

Document d'organisation :
https://docs.google.com/document/d/1pV66-_grl-AKpCXYLLRne3QlC4mrltYHJGQhSN1PmxU/edit#heading=h.knvdzsc0fhiq

Archives :
https://docs.google.com/document/d/1edIJvYdL4zctuAWvqBicaBCWEzKQpjQvc6_GrE_qGxo/edit#heading=h.71kcowjy648j