

|  |
| --- |
|  |
| Cahier de recettes |
| Projet algorithmique semestre 4 |
| GEIGUER Pierre | FERNANDO Théo |LAMBALLAIS Hugo |

|  |  |
| --- | --- |
|  | Décoratif |
| Introduction |
| Ce document vise à fournir une liste complète de tests pour valider les exigences du projet final. Les tests ont été élaborés pour garantir que toutes les exigences du projet sont satisfaites.  Chaque test est décrit avec précision et est accompagné d'instructions détaillées pour permettre une reproduction facile. Les tests sont organisés de manière à refléter la structure et les exigences du projet final.  Ce cahier de recettes de projet est un outil indispensable pour l'équipe de développement et les testeurs, car il garantit que le produit final est entièrement testé et validé avant d'être livré. |
|  |
|  |

|  |  |
| --- | --- |
|  | Décoratif |
| Fonctionalitées et testsExigences d’initialisation :  1. Le programme initialisera la fenêtre   La fenêtre est initialisée grâce à une fonction intégrée dans SDL2.   1. En cas d’erreur d’initialisation un message sera affiché   Les erreurs sont renvoyée grâce à “SDL\_GetError()”.   1. Le programme créera un pointeur de rendu graphique   Le renderer est créé dans la même fonction de création de la fenêtre.   1. En cas d’erreur de création un message sera affiché   Le programme vérifie si le pointeur est NULL.  Test : Vérifié grâce à une condition.   1. Le programme créera des textures lors d’un niveau   La création des textures se fait grâce à une fonction “loadTexturePNG()” qui prend en paramètre le renderer, les positions de l’objet, la texture de l’objet. Cette fonction retournera la texture et modifiera les positions en y mettant la taille de l’objet.   1. Une surface est initialisée avec les données de l’image correspondantes   On va lire une image et mettre les données dans une surface.  Test : Vérifié grâce à une condition lors de la récupération.   1. En cas d’erreur d’écriture un message sera affiché   La fonction vérifiera également si le pointeur de la surface est NULL et renverra un message d’erreur.  Test : Vérifié grâce à une condition.   1. Une texture sera créée à partir de la surface   On va lire la surface et mettre les données sous forme de textures.  Test : Vérifié grâce à une condition lors de la création.   1. En cas d’erreur d’écriture un message sera affiché   La fonction vérifiera si le pointeur est null et renverra un message d‘erreur.  Test : Vérifié après l’écriture pour vérifier si la texture est NULL   1. Le programme initialisera l’appel de création de texte   Utilisation de TTF font.  Test : Vérifié grâce aux fonctions SDL intégrées.   1. En cas d’erreur d’initialisation un message sera affiché   Test : Vérifié grâce à une condition sur le pointeur de la police qui est différent de NULL.   1. Le programme lira une police d’écriture   Test : Vérifié grâce à une fonction intégrée de SDL.   1. En cas d’erreur de lecture un message sera affiché   Le programme renverra une erreur grace à la fonction intégrée de “TTF\_getError()”. Exigences du système :  1. Le système sera un personnage   Le système est représenté par une structure dans l’implémentation du code, cette structure contient les informations du personnage comme sa vitesse, son saut, ses textures et ses positions.   1. Le système sera un astronaute   Une texture d’astronaute a été initialisée dans la structure du personnage grâce à une fonction “setAstronaute”   1. Le système devra pouvoir se déplacer de gauche à droite   La gestion des déplacements se fait par la détection des touches enfoncées avec les fonctions de SDL2.  Si la flèche droite est enfoncée, le déplacement est maintenu à 5 pixels par itération.  Si la flèche gauche est enfoncée, le déplacement est maintenu à -5 pixels par itération.  Si aucune touche est enfoncée alors la vitesse est de 0.  Le test a été effectué an affichant le personnage sur l’écran et en observant s’il y avait déplacement latéral et du bon nombre de pixel.  Test : Vérifié.   1. Le système devra pouvoir sauter   La gestion du saut de fait également par la détection de la touche “Flèche du haut” enfoncée.  Lorsque cette touche est enfoncée, le système se déplacera sur les coordonnées en y et en x et retombera automatiquement grâce à l’implémentation d’une variable gravité qui réduira la position y jusqu’à ce le système retombe sur un objet compatible.  Test : Vérifié.   1. Le système ne devra pas sortir de l’écran   Les positions du système sont comparées au position des bords de l’écran pour que celui-ci n’en sorte pas.  Le système prendra automatiquement la position maximale autorisée.  Test : Vérifié.   1. Le système devra évoluer sur une image de fond   Le fond d’écran se déplace lorsque le personnage atteint les trois quart de l’écran. Il y a donc deux fonds d’écrans l’un à la suite de l’autre qui avance en permanence et qui stoppe le personnage aux trois quarts de l’écran.  Test : Vérifié grâce à une fonction qui effectue des vérifications sur les coordonnées des fonds d’écran.   1. Le système pourra se déplacer sur un sol   Les sols sont des textures ressemblant à un sol d’autre planète, ils sont créés grâce aux fonctions loadTexturePNG.  Test : Vérifié grâce à une condition sur les pointeurs qui sont différents de NULL.   1. Le système sera soumis à des collisions   Une fonction “collision” a été mise en place pour vérifier dans un premier temps s’il y a une collision entre le système et un objet en utilisant “SDL\_IntersectRect”. Dans un second temps, elle va vérifier de quel côté la collision a eu lieu pour pouvoir placer le système à gauche, à droite ou sur l’objet.  Une fonction “collision\_enemis” a été implémentée pour gérer les collisions du système avec un ennemi.  Si la collision est latérale alors la partie est perdu, si la collision est au-dessus alors l’ennemi disparait.  Test : Vérifié grâce à l’affichage des coordonnées des collisions qui correspondent à la collision visible.   1. Le système devra évoluer dans plusieurs niveaux de type platformer   Deux niveaux ont été implémentés dont un de présentation.   1. Le système rencontrera des ennemis   Les ennemis sont implémentés de la même façon que l’astronaute sauf qu’ils possèdent une structure Ennemi contenant les textures et les positions. Ces ennemis se déplacent lorsque le personnage est à proximité.  Test : Vérifié en s’approchant de l’ennemi et en affichant ses coordonnées, sa vitesse et en vérifiant sa création grâce à des conditions.   1. Les ennemis feront arrêter la partie en cas de collision avec le système   Des monstres ont été implémentés, le système doit donc éviter les collisions latérales sous peine de défaite et doit éliminer l’ennemi avec une collision supérieur.  Les ennemis sont mobiles.  Le test a été effectué en affichant un ennemi à l’écran pour lequel le système est soumis à plusieurs collisions.   1. Le système pourra finir la partie en montant dans un vaisseau   Un vaisseau de fin de niveau a été implémenté, il possède sa propre structure vaisseau qui contient toutes les textures et les positions pour faire une animation d’envol en modifiant les coordonnées et en alternant les textures pour simuler le démarrage.  Test : Vérifié en affichant les coordonnées de déplacement souhaitées et en les comparant à celles renseignées dans le code. Exigences de sauvegarde :  1. L’utilisateur devra pouvoir sauvegarder sa progression dans un fichier texte 2. Le système enregistrera le nombre de morts, les pièces, le niveau, les personnages   Test : Vérifié en comparant le nombre d’éléments sauvegardés au nombre d’élément à sauvegarder dans le code.   1. Le système lira le fichier texte   Test : Vérifié grâce à une condition.   1. Le texte sera des nombres au format décimal   Test : Vérifié, tous les compteurs sont au format décimal.   1. Les nombres du fichier seront séparés par des retours à la ligne et des espaces   Test : Vérifié dans le code et dans le fichier de sauvegarde.   1. En cas d’erreur de lecture un message sera affiché   Test : Vérifié grâce à des condition de lecture. Exigences des menus :  1. L’utilisateur aura accès à des menus   Test : Vérifié en appuyant sur les boutons concernés et en affichant si l’utilisateur entre bien dans les boucles correspondantes.   1. Les menus proposeront différents choix liés au système   Test : Vérifié, plusieurs choix sont affichés   1. Les menus vérifieront les coordonnées de la souris pour connaitre le choix   Test : Vérifié en affichant les coordonnées de la souris et en la déplaçant.   1. Les choix changeront de couleur lors du survol   Test : Vérifié au niveau visuel et en affichant grâce à une condition s’il y a eu un changement.   1. Les menus se fermeront lors d’un clic gauche sur un choix   Test : Vérifié au niveau visuel et en affichant le choix sélectionné. Exigences de la progression :  1. Le programme affichera le nombre de morts   Un compteur de morts a été implémenté et celui-ci est affiché grâce à une fonction TTF qui écrit du texte sur une texture pour ensuite l’afficher à l’écran. Le nombre de mort incrémente lorsque le personnage entre en collision avec un ennemi.  Test : Vérifié au niveau visuel et en affichant le nombre de mort enregistré dans le programme avec celui affiché.   1. Le programme affichera le meilleur temps du niveau   Le meilleur temps sera sauvegardé à la fin du niveau s’il est plus faible que celui d’avant. Il est ensuite affiché à l’écran de la même façon que le nombre de morts.  Test : Vérifié en affichant les deux temps et en vérifiant si le plus petit est bien enregistré.   1. Le programme affichera le temps passé dans le niveau   Le temps est affiché en partie, il prend en compte le temps actuel grâce à SDL\_GetTicks() et en y enlevant le temps de commencement du niveau. Il est ensuite affiché de la même manière que le nombre de morts.  Test : Vérifié en affichant le temps dans la console et en le comparant à celui affiché sur la fenêtre du jeu.   1. Le programme affichera le nombre de pièces récoltées   Le nombre de pièces récoltées est calculé à la fin du niveau, un nombre de pièces est fixé à 300 et à la fin on y enlève le temps passé dans le niveau. Au plus ce temps est faible, au plus le nombre de pièce sera important.  Ce nombre de pièce sera ensuite affiché de la même manière que le nombre de mort.  Test : Vérifié en affichant le nombre de pièce dans la console et en le comparant à celui affiché dans le fenêtre de jeu. |
|  |  |