



Projet Pluridisciplinaire d'Informatique Intégrative 2
PPII 2
1A TELECOM Nancy - 2024

Rapport de projet PPII 2

\debug_mode

Groupe de projet

Olivia AING (cheffe de projet)
Chloé BONINN
Lise MANIÈRE
Sylvie SIDLER
Zoé VERNICOS

Professeurs référents

Christophe BOUTHIER
Olivier FESTOR
Gerald OSTER
Pierre-Etienne MOREAU

Sommaire

1 Introduction

2 Conception

- 2.1 Choix du concept et style de jeu
- 2.2 Gestion de projet

3 Réalisation

- 3.1 Menu start
- 3.2 Sauvegardes
- 3.3 Gestion des sons
- 3.4 Déplacements joueur
- 3.5 Carte
- 3.6 Collisions
- 3.7 A★
- 3.8 Terminal
- 3.9 makefile
- 3.10 Dialogues
- 3.11 Graphismes & Sons

4 Bilans de projet

- 4.1 Bilan de groupe
- 4.2 Bilans individuels

5 Annexes

- 5.1 Sources
- 5.2 Preuves
- 5.3 Game Design Document
- 5.4 Comptes rendus de réunion

1 Introduction

Les consignes pour ce projet étaient d'utiliser le langage C et SDL2 afin de créer un jeu vidéo inspiré des jeux des années 80.

Après lecture du sujet de PPII2, notre groupe a commencé par faire un brainstorming afin de savoir dans quelle direction se lancer. Nous avons cherché des idées dans les jeux que nous connaissons et avons notamment évoqué les jeux *Zelda* en 2D et *There is no game*, ce qui nous a aidé à trouver l'inspiration. C'est ainsi que nous en sommes arrivés à l'idée de notre jeu vidéo : `\debug_mode`.

Nous commencerons dans notre rapport par présenter le concept du jeu, avant de présenter nos choix d'implémentations et les problèmes que nous avons rencontrés, et pour finir nous ferons un bilan individuel et global.

2 Conception

2.1 Choix du concept et style de jeu

Très rapidement, nous avons décidé de créer un RPG. Il s'agissait d'un classique, plutôt intemporel, et qui avait des codes assez clairs pour que l'on puisse se lancer assez rapidement dans la planification de l'évolution du jeu.

Une autre feature qui est arrivée très rapidement, voire dès le début, était l'idée d'un "terminal", et du "\debug_mode". Nous voulions offrir à nos joueurs une sorte d'introduction ludique au monde de l'informatique. Quant aux implications de l'accessibilité d'un tel mode de jeu, elle est venue avec l'idée du monde incomplet. Après cela, le personnage principal, Kiki, a été créé, et pour pouvoir aider le joueur dans sa découverte du jeu, Tutoriel l'a rapidement suivie. Nous avons également partiellement fait un Game Design Document (GDD), pour poser un peu mieux les différentes idées que nous avions (voir annexe), en plus de nous donner une première idée de la timeline possible du projet. Il s'agit d'un document qui est utilisé dans l'industrie du jeu vidéo, spécifiquement pour mettre sur papier les idées principales du jeu, auxquelles on n'aurait pas forcément pensé.

Le projet \debug_mode était né.

Au départ, nous voulions, de la même manière que pour par exemple *Zelda*, avoir une vue à la troisième personne, en trois quarts. Toutefois, en approfondissant un peu plus nos recherches, et après avoir été confrontées à certains problèmes techniques ou d'implémentation, nous avons porté notre choix sur une vue de dessus, à la troisième personne.

Le jeu, tout au long de sa création, a évolué, et les mécaniques avec, pour pouvoir s'adapter à ce que nous étions capable de faire à un moment donné. Après tout, nous avons un temps très limité, et il a fallu faire des concessions, pour pouvoir rendre un jeu avec les mécaniques clefs que nous voulions créer.

Dans les idées que nous n'avions pas pu implémenter, il y avait notamment d'autres commandes, qui permettaient d'interagir avec des PNJ (Personnages Non Joueurs), des ennemis, ou des boss, comme par exemple un boss chaussure, qui aurait essayé d'écraser Kiki. Nous avons préféré nous focaliser sur les mécaniques impliquant directement le \debug_mode, et donc la résolution de puzzle.

2.2 Gestion de projet

Une fois le concept validé, nous nous sommes réunies, pour pouvoir définir ce que nous devons créer, au minimum, pour que le jeu remplisse nos attentes minimales. Pour cela, nous avons encore fait une réunion de brainstorming, mais plus autour des nécessités pour le jeu. Nous avons donc créé ensemble un Work Breakdown Structure, pour nous aider dans cette tâche.

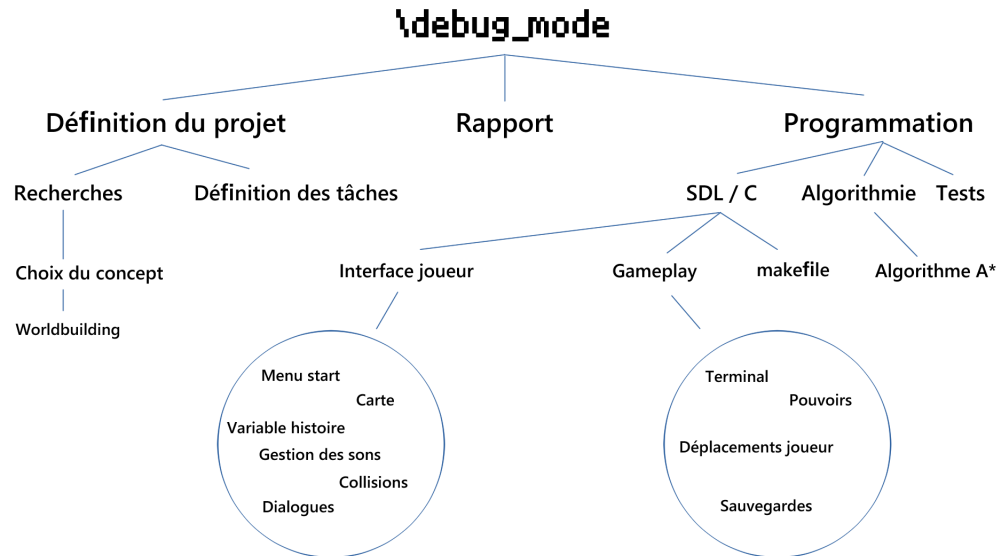


FIGURE 1 – Work Breakdown Structure (WBS)

Nous avons ensuite établi une matrice RACI, dont les membres avec l'autorité sur les tâches n'ont pas changé, mais dont les réalisateurs et conseillers ont évolué au fur et à mesure du projet, selon les besoins. Il fallait également faire en sorte que les tâches soient réparties aussi uniformément que possible, puisque créer un jeu, même de petite envergure, allait nous prendre du temps, et beaucoup d'apprentissage. Pour la répartition de ces tâches, puisque chacune d'entre elles était assez conséquentes, nous sommes parties, comme pour le PPII 1, sur les tâches que nous voulions accomplir. Après tout, rien ne valait la motivation d'accomplir une tâche que nous avons choisie, plutôt que celle d'une tâche imposée !

MATRICE RACI	Acteurs				
Etapes	Sylvie Sidler	Zoé Vernicos	Chloé Bonnin	Lise Manière	Olivia Aing
Définition du projet	R	R	R	R	RA
SDL / C					
Menu start	RA	I	I	I	I
Gestion des sons	RA	I	I	I	I
Déplacement joueur	I	RA	I	I	I
Carte	I	I	I	RA	I
Terminal	C	RA	I	I	R
Dialogues	I	I	I	I	RA
Gestion de la variable histoire	I	I	I	I	RA
Fonctions des pouvoirs	I	I	RA	I	C
Gestion des sauvegardes	RA	I	C	I	C
Collisions	I	C	R	RA	I
makefile	C	RA	I	I	C
Algorithmie					
Algorithme A*	I	I	RA	I	I
Construction de l'univers					
Graphismes	R	C	C	C	RA
Musiques & Sons	I	I	I	I	RA
Ecriture de l'histoire	R	R	R	R	RA
Tests	R	R	R	R	RA
Rapport	R	R	R	R	RA

Légende	
R	Réalise
A	Autorité
C	Conseil
I	Informé

FIGURE 2 – Matrice RACI

Nous avons également fait un diagramme FFOM (Forces internes, Faiblesses internes, Opportunités externes, Menaces externes) pour pouvoir gérer au maximum les difficultés que nous allons ultimement rencontrer au cours du projet. Un des exemples pour pallier à la SDL (sans parler des problèmes techniques liés à la bibliothèque elle-même), était de trouver des sites pour apprendre à utiliser la bibliothèque. (voir annexe)

Forces	Faiblesses
Expérience d'équipe	SDL = Nouveau langage
Motivation	RPG = long à faire
Communication	
Opportunités	Menaces
RPG = classique du jeu vidéo	Partiels
Réseau (camarades, professeurs)	Temps

FIGURE 3 – Diagramme FFOM (Forces, Faiblesses, Opportunités, Menaces)

3 Réalisation

3.1 Menu Start

Le menu Start a été implantée à part avec sa propre boucle d'évènements.

À l'ouverture, le joueur peut démarrer une nouvelle partie ou sélectionner une sauvegarde à charger (cf. 3.7 Sauvegardes). Si le joueur choisit de charger une sauvegarde vide, le jeu lance une nouvelle partie.

Le menu option permet de choisir dans quelle sauvegarde sera sauvegardée la partie à lancer, par défaut la sauvegarde n°0 est utilisée. Les crédits du jeu sont également disponible, affichant une courte présentation du jeu et du projet ainsi que les membres du groupe.

Ayant eu des problèmes avec `SDL_TTF`, j'ai choisi d'afficher le menu de manière similaire aux sprites : avec un fichier contenant les 6 images de fond d'écran changeant en fonction de la fenêtre. L'absence de structure pour des boutons cliquables m'a posé problème, ils sont donc définis par des rectangles `SDL_Rect` définissant leurs tailles et positions. Lors d'un clic, il est donc vérifié si le clic a été fait dans la surface du bouton avant de réaliser les actions correspondantes.

3.2 Sauvegardes

La sauvegarde est une structure C permettant de stocker différentes informations :

Numéro de sauvegarde : un booléen définissant dans quel fichier enregistrer la sauvegarde : `sauvegarde0.txt` ou `sauvegarde1.txt`

Avancement : un entier décrivant l'avancement du joueur dans l'histoire du jeu et des cutscenes.

CarteId : un entier correspondant au numéro de la carte dans laquelle se trouve Kiki au moment de la sauvegarde.

Positions x et y : des entiers correspondant aux positions x et y de Kiki dans la salle `CarteId`.

idVisites : une liste des entiers `CarteId` représentant les salles visitées par Kiki.

les sauvegardes sont implémentées autour de 2 fonctions principales permettant de charger ou enregistrer une sauvegarde depuis/dans un fichier texte. Il a été discuté d'utiliser des fichiers binaires pour enregistrer les sauvegardes mais pour des raisons pratiques, il s'agit de fichiers en `.txt` pour faciliter le debugging en changeant directement les valeurs nécessaires dans les fichiers.

Lors du menu Start, si le joueur choisit de charger une sauvegarde, le jeu s'enregistrera sur ce numéro de sauvegarde. Si la sauvegarde est "vide", elle correspond à un l'état du jeu lors d'une nouvelle partie. Le jeu reprend alors au niveau de la sauvegarde avec Kiki retournant à sa position du début de niveau.

Dans le cas où le joueur lance un nouveau jeu, une nouvelle sauvegarde à vide est créée

et elle s'enregistrera automatiquement sur la n°0.
En quittant le jeu, la partie actuelle est automatiquement sauvegardée.

3.3 Sons

Il existe 2 types de sons différents dans le jeu : la musique de fond et les effets sonores, qui doivent pouvoir être joués simultanément.

La musique de fond est différente entre le menu Start et le jeu, il faut donc arrêter la musique en sortant du menu Start puis lancer la nouvelle musique du jeu. Les fonctions de `sounds.c` servent à charger les fichiers audios dans les structures de `SDL_Mixer` : `Mix_Music` et `Mix_Chunk` pour la musique et effets respectivement.

Pour que la musique ne soit pas stoppée par les effets sonores, ils doivent être joués dans un channel sonore différent. Il y a 8 channels disponibles par défaut dans `SDL_Mixer` donc la fonction jouant les effets sonores utilise le premier channel libre sans couper la musique de fond. Une option dans la fonction de `SDL_Mixer` jouant la musique permet de le faire en continu dans le premier channel sans avoir à rappeler cette fonction.

3.4 Déplacements joueur

Problèmes rencontrés et résolution :

affichage du sprite : l'affichage du sprite a rencontré plusieurs problèmes au départ il ne marchait pas à cause d'une erreur dans la texture, dont l'origine était le fait que l'initialisation n'était pas faite à l'intérieur du main.

découpage et affichage du sprite : le deuxième problème de l'affichage du sprite était de bien découper le spritesheet pour afficher un carré de bonne taille et donc d'afficher correctement le carré. Pour ce faire, j'ai utilisé deux `SDL_rect` : un pour le rectangle de destination et un pour celui du sprite qui découpe le spritesheet.

keyboard input : pour faire déplacer le sprite nous avons décidé qu'il fallait que l'on puisse le faire en utilisant les flèches directionnelles mais aussi les touches ZQSD ou WASD dépendant du clavier, pour ce faire j'ai utilisé `SDL_scancode` qui permet de récupérer la position de la touche dans le clavier et permet de récupérer la touche voulue peu importe le clavier.

orientation du sprite : au bout d'un moment le sprite se déplace mais l'image affichée ne se déplace pas (le sprite bouge mais toujours comme s'il était dirigé vers le bas). pour ce faire j'ai créé une structure permettant de savoir si le sprite se déplaçait dans une des quatre directions et si oui, avec un test on change de ligne dans le sprite sheet.

3.5 Carte

Le choix a été fait de créer des cartes en faisant une matrice carré composée de 0 et de 1 (les 1 correspondant au mur. Du fait d'un pouvoir `debug_mode`, nous avons ajouté des murs non franchissables en debug mode représentés par des -1. De plus des interrupteurs permettant d'ajouter ou de supprimer des murs (variables <-1) et d'autres permettant de passer d'un niveau à un autre (variables >1) ont été créés. Pour passer d'une carte à une autre, nous avons eu besoin de faire une liste de cartes : une liste de `int**` or nos cartes sont implémentées en `int[][]`. Il a donc fallut que nous implémentions une fonction transformant nos cartes `int[][]` en `int**`. Cela nous a donc permis de créer 2 listes (au travers de 2 fonctions) contenant pour l'une toutes les cartes des niveaux et pour l'autre les dérivées des cartes principales avec des murs en plus ou en moins suivant le niveau. Nous avons créés une variable `carte` dans le fichier `main` correspondant à la carte dans laquelle se trouve Kiki et une variable `dans_carte` donnant le numéro de la carte.

Nous avons créé des fonctions permettant de libérer la mémoire utilisée par chaque carte qui sont utilisées lorsque le jeu est fermé.

Enfin, plusieurs fonctions `afficher_carte` permettent afficher chaque carte en fonction de sa spécificité (la `carte0` ne doit pas afficher les mur, la `carte1` ne possède pas de couleurs, la `carte2` de possède que la couleur bleu et la `carte3` possède les couleurs bleu et rouge). Les murs sont gris clairs pour les murs traversable avec `debug_mode`, gris foncé pour les murs non traversables en `debug_mode`, blanc pour les interrupteurs avant de posséder leur couleur, bleu pour les interrupteurs permettant de changer de niveau et rouge pour les interrupteurs permettant de créer ou de supprimer des murs.

3.6 Collision

Il y a 3 types de collisions : celles avec les murs, celles avec les interrupteurs de changement de niveau et celles avec les interrupteurs créant ou supprimant des murs.

Elles sont toutes gérées de la même manière : on parcourt la matrice de la carte en cours, si le sprite de Kiki a une collision avec un mur ou un interrupteur, il y a un changement de position de Kiki. Dans le cas où Kiki rentre en collision avec un mur, elle prend en position sa position précédente (sauvegardée en début de boucle). Le test pour vérifier qu'il y a une collision est réalisé avec `SDL_HasIntersection` qui permet de tester l'intersection entre deux rectangles (ici un correspondant à la hitbox de Kiki et l'un correspondant à une case de la carte).

Dans le cas où Kiki arrive sur un interrupteur, la carte est changée et si c'est un interrupteur de changement de niveau, elle prend en position les positions d'initialisation de la salle suivante (positions enregistrées dans un tableau en début de fichier).

De plus dans le cas où l'on se trouve en `\debug_mode`, seules les collisions avec les cases en -1 de la matrice de la carte sont prises en compte.

3.7 A★

Nous avons décidé de créer pour le joueur un pouvoir lui permettant d'afficher une carte sur laquelle le trajet pour atteindre la prochaine salle depuis sa position actuelle serait tracé. Afin de trouver le chemin à tracer, nous avons décidé d'utiliser l'algorithme A★. L'algorithme aurait également pu être utilisé pour vérifier l'existence d'un chemin vers la sortie si nous avions créé des cartes aléatoirement.

Nous avons choisi d'implémenter l'algorithme A★ car il permet un compromis entre rapidité d'exécution et exactitude de la réponse. En effet, A★ n'est pas un algorithme exact mais il permet, en faisant des choix basés sur des estimations des parcours, de trouver un chemin assez performant et en un temps rapide. La rapidité de l'algorithme choisi était importante puisque lors de l'utilisation de la carte, le chemin optimal est recalculé à chaque déplacement du personnage.

Pour implémenter A★, nous avons créé une structure de files prioritaires. Les noeuds des files prioritaires permettent de sauvegarder une position sur la carte et d'y associer une estimation de la longueur d'un chemin passant par cette position (fonction f). Ils conservent également le noeud d'où ils viennent comme parent et sont triés par ordre croissant de leur évaluation par f . La distance est estimée grâce à la distance de Manhattan.

La fonction `add` permet d'ajouter un nouveau noeud à la bonne place dans la file prioritaire. Elle duplique le noeud ajouté afin qu'il n'y ait pas de conflits entre les parents des noeuds et les noeuds présents dans la file prioritaire.

Des tests unitaires ont été créés pour l'implémentation des files prioritaires et de A★.

Pour afficher une carte avec le tracé du chemin trouvé par A★, nous avons implémenté une fonction renvoyant une texture afin de pouvoir la placer en taille réduite à côté de la carte principale. De plus, afin que le temps d'exécution reste raisonnable et que la vitesse de déplacement du personnage ne soit pas trop ralentie, le chemin n'est pas recalculé à chaque itération de la boucle principale mais seulement au bout d'une dizaine d'itérations.

3.8 Terminal

Le terminal est un un élément interactif de notre jeu qui permet d'activer différents pouvoir du jeu à savoir :

- `\close` ou `\c` qui permettent de fermer le terminal ;
- `\map` qui permet d'ouvrir la carte en parallèle de l'espace de jeu et aussi le chemin à parcourir pour arriver à un but : sortie ou interrupteur ;
- `\close_map` qui permet de fermer la carte ;
- `\save` qui permet au joueur de sauvegarder sa progression dans le jeu ;
- `\debug_mode` un mode de jeu qui permet de passer à travers les murs en désactivant les collisions ;
- `\normal_mode` qui permet de désactiver le `debug_mode`.

Au cours de l'élaboration du terminal j'ai rencontré plusieurs problème car le terminal devait permettre au joueur d'écrire pour activer les pouvoirs, donc de pouvoir aussi supprimer du texte en cas de faute de frappe, puis il fallait que le fait de faire cliquer sur ENTREE permette d'activer les pouvoirs.

Pour la partie écriture, j'ai choisi d'utiliser un `char*` pour pouvoir y rajouter des éléments au fur et à mesure de la saisie du joueur. Ensuite il fallait que dès qu'un joueur appuie sur une touche, la lettre saisie soit affichée à l'écran. j'ai donc utilisé l'événement `SDL_TEXTINPUT` qui permettait de récupérer les touches du clavier, et de le mettre dans le `char*` puis en créant une nouvelle texture avec TTF, on affiche le texte dans le renderer.

La prochaine étape a été de faire en sorte que le texte s'efface et qui le fasse aussi sur le terminal (visuel) ainsi en prenant en compte lorsque la touche `backspace` était pressée, on supprime les éléments 1 par 1 dans le `char*` et on efface petit à petit en recouvrant la zone de texte avant de ré-afficher le nouveau texte par dessus.

Enfin la dernière étape pour que le terminal fonctionne était que lorsque les noms des pouvoirs sont saisis et qu'on appuie sur ENTREE que ceux-ci soient activés. J'ai donc utilisé une vérification lettre par lettre pour vérifier quel pouvoir était appelé, ce qui entraîne donc soit un appel à une fonction, soit le changement d'une variable.

3.9 makefile

Le makefile devait remplir deux conditions :

make : on a fait en sorte que le jeu compile lorsque on fait `make` dans le terminal.

make test : on a fait en sorte que les tests compilent lorsqu'on fait `make test` et qu'ils se lancent

3.10 Dialogues

Les dialogues devaient être implémentés dans des boucles d'événements à part, afin de pouvoir faire des cinématiques.

La première difficulté a été de faire en sorte d'afficher correctement les boîtes de dialogues.

De la même manière que pour le terminal, il fallait que la boîte ait un fond opaque, pour que l'on puisse lire de manière lisible le texte apparaissant au dessus. Il fallait également deux versions de chaque boîte, qui sont trouvables dans "window_display.h", permettant d'avoir une version en noir et blanc, et une autre avec le bleu.

Ensuite, il fallait utiliser la bibliothèque d'images de la SDL, une bibliothèque à part, pour pouvoir faire apparaître le cadre, et les portraits. On importe l'image que l'on veut faire apparaître en tant que surface, puis on la transforme en texture, pour pouvoir la faire apparaître sur la fenêtre de jeu, tout en la mettant au bon endroit, et surtout à la bonne taille.

Le problème suivant a été de faire fonctionner TTF, une autre bibliothèque de la SDL supportant les polices d'écritures.

De la même manière que pour les images, on fait apparaître le texte dans une surface, avec `TTF_RenderUTF8_Solid_Wrapped`, une fonction qui permet de créer une surface à largeur limitée, que l'on peut ensuite transformer en texture. Pour que le texte n'apparaisse pas étiré, il fallait adapter la hauteur et la largeur de la zone d'impression aux dimensions de la surface créée par le texte, en utilisant `SDL_QueryTexture`. Puis, comme pour le cadre et les portraits, on peut faire apparaître le texte au bon endroit, et à la bonne taille.

Nous avons eu l'occasion, avec l'utilisation de TTF, de voir qu'il fallait utiliser un autre type de rendu pour le renderer sur les ordinateurs avec MacOS (`SDL_RENDERER_SOFTWARE` au lieu de `SDL_RENDERER_ACCELERATED`). Autrement, la bibliothèque TTF faisait apparaître des erreurs, et fermait le jeu.

Une des difficultés majeure des dialogues était bien évidemment leur écriture : il n'y avait que deux personnages, Kiki et Tutoriel, mais il fallait faire en sorte qu'ils aient tous les deux leur propre personnalité, et qu'elle se fasse ressentir à travers le peu de dialogues, et de leur portraits.

D'un côté, Kiki, qui représente la protagoniste du jeu, devait être plutôt innocente et optimiste, suivant simplement son guide, Tutoriel, à travers le monde inexistant dans lequel elle était. Ses émotions devaient retranscrire une certaine candeur, découvrant le jeu pour la première fois, comme le joueur. Le parallèle est aussi retranscrit dans ses compétences propres sont ce qui influe directement sur le jeu : comme le joueur, elle est une actrice clef de l'histoire.

De l'autre, on avait donc Tutoriel, qui lui est plus un "spectateur" du jeu. Il essaie de reconstruire le monde dans lequel il est censé apparaître, et d'aider Kiki au possible

dans sa quête. Contrairement à elle, il en sait beaucoup sur le monde dans lequel il a été créé, et n'hésite pas à casser le 4^e mur, pour s'adresser en partie directement au joueur. Ses compétences se limitent presque à la transmission d'informations, à la fois à Kiki et au joueur, puisque même s'il est capable de faire du "code diving" (donc de "voir" ce qu'il y a derrière le jeu), il est incapable d'y modifier quoi que ce soit. C'était aussi l'occasion de faire en sorte que Tutoriel porte bien son nom : comme un classique tutoriel de jeu vidéo, il explique les mécaniques qu'il y a derrière le jeu, pour que le joueur puisse les exploiter.

Enfin, dans les fonctionnalités supplémentaires qu'il fallait rajouter, il y avait :

passer les dialogues :

Elle permettait, entre autres, d'aller plus vite dans les tests, mais surtout de permettre aux joueurs éventuellement plus aguerris de sauter les explications de bases, qui sont longues, et relativement fastidieuses. Pour pousser le vice jusqu'au bout, on a permis le saut de toutes les cinématiques.

Le saut de cinématique a été implémenté grâce au maintien d'une touche pendant un certain temps, comptabilisé par un compteur, pour être sûr que le joueur veut bel et bien sauter la cinématique.

fermer la fenêtre en plein milieu d'une cinématique :

Il fallait faire en sorte de pouvoir fermer le jeu en plein milieu d'une cinématique si nécessaire, en pensant au joueur qui devait éventuellement partir en plein milieu. Pour cela, on compare l'avancée de l'histoire avant la cinématique, gardée dans une variable *vh*, et celle d'après, dans *sauvegarde->avancement*. Si jamais l'avancée n'avait pas changé, alors le joueur a décidé de fermer la fenêtre en pleine cinématique, et on envoie le signal pour fermer la fenêtre de jeu.

des choix avec des conséquences :

L'incrémentation de la variable histoire dépendait de si l'on avait été "gentil" ou non avec Tutoriel. Ainsi, ses dialogues changeaient, selon si on sautait les dialogues ou si on les lisait jusqu'au bout. On donnait ainsi à Tutoriel la capacité de se "souvenir" de ce qu'il se passait en dehors du jeu.

Pour que l'avancement de l'histoire soit toujours correct malgré ce choix, il fallait à chaque fois avoir deux variables histoires correspondant à un même événement : une paire si on avait lu les dialogues jusqu'au bout, et une impaire le cas échéant. L'incrémentation de cette variable se faisait donc selon la variable histoire donnée en entrée, et augmentait de 1, 2 ou 3 selon le choix du joueur. L'augmentation correcte de cette variable a été vérifiée par récurrence (voir annexe).

demander l'aide de Tutoriel :

Pas tous les joueurs sont aguerris, aussi avoir accès à un rappel de certaines commandes peut être utile. Il fallait donc au moins leur donner accès à un endroit leur rappelant par exemple comment ouvrir le terminal.

3.11 Graphismes & sons

Les graphismes et le sound design sont des éléments servant à construire le monde d'un jeu vidéo. Ils soutiennent, dans le cas de ce RPG, l'histoire que l'on raconte dans le jeu.

Pour les musiques de fond, que ce soit celle de l'écran titre ou celle du jeu, elles ont été choisies dans la banque de musique de RPG Maker, pour coller à une ambiance plutôt douce, et calme, pour se mettre dans un bon état afin de résoudre les différents puzzles. Quelques sons ont été ajoutés, en créant des signaux sur Audacity, et en les combinant. Par exemple, les sons des dialogues pour Kiki et Tutoriel (qui n'ont malheureusement pas pu être utilisés à cause de problèmes techniques), sont des signaux de forme différente, pour donner un son plus doux à Kiki qu'à Tutoriel, et ayant des fréquences qui ne sont pas semblables non plus, pour que ces deux personnages aient leur propre "voix".

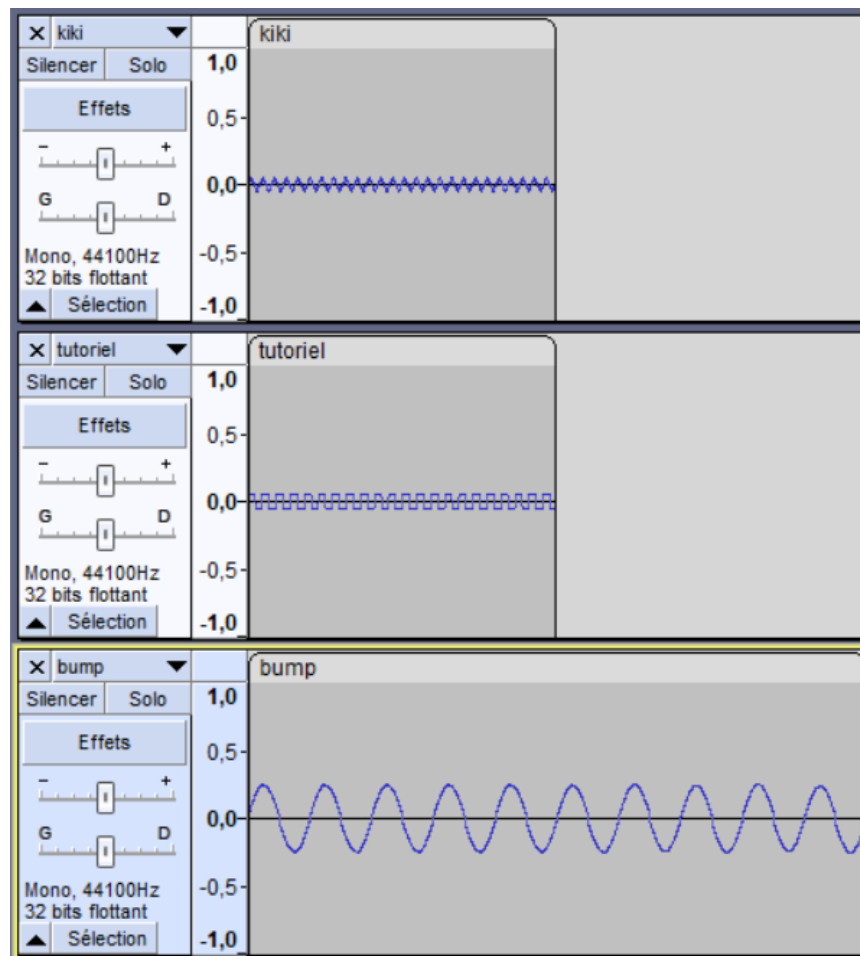


FIGURE 4 – Audacity - Son de Kiki, son de Tutoriel, et son du bumper de collision

Au niveau visuel, il y avait plusieurs éléments à choisir, et à prendre en compte. Tout d’abord, pour coller au style graphique des jeux des années 80, il fallait forcément se diriger vers du pixel art. Alors, avant même de commencer à créer le moindre asset, à cause de la nature de RPG de notre jeu, il fallait choisir une police d’écriture lisible et qui allait avec le reste des assets graphiques. Pour cela, on a fait une sélection de polices libres de droit que nous pouvions utiliser, avant d’en choisir une qui permettait de faire ce que l’on voulait.



FIGURE 5 – Liste non exhaustive de polices d’écritures possibles

Enfin, pour tous les assets graphiques restants, il a fallu apprendre à faire du pixel art. Il y a eu des tests, et des évolutions, au niveau des dessins. Certains n'ont pas été utilisés, par manque de temps, ou parce qu'ils ne correspondaient plus au jeu que nous voulions créer. La difficulté majeure dans tous les cas était la limitation de palette, puisqu'au début du jeu, et donc tout au long de la démo, nous n'avions accès qu'au noir et au blanc. L'intégration de détails était donc assez compliquée à cause de cela.

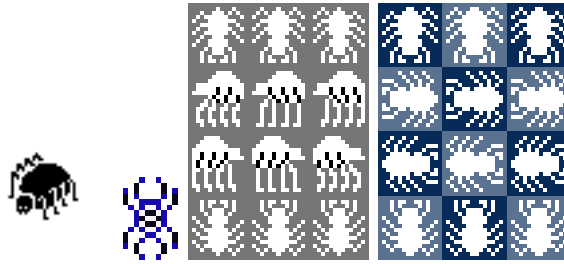


FIGURE 6 – Evolution du sprite de Kiki



FIGURE 7 – Tileset

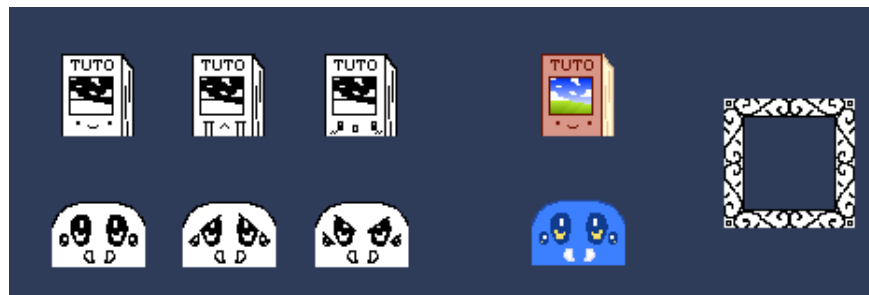


FIGURE 8 – Ensemble des assets de dialogue

4 Bilans de projet

4.1 Bilan de groupe

Du point de vue du travail d'équipe, le groupe étant composé des mêmes personnes que lors du PPII 1, nous étions déjà habituées à travailler ensemble, ce qui nous a permis de gagner du temps pour l'organisation. La bonne entente au sein du groupe et la création d'un serveur discord pour nous organiser et communiquer nous a permis de rester motivées et à jour sur nos avancées. Nous nous sommes également régulièrement réunies pour faire des réunions et des séances de travail en groupe afin d'éviter l'effet tunnel et d'avancer plus efficacement. La gestion de projet nous a permis de travailler régulièrement et facilement grâce à des tâches claires.

Du point de vue du projet, nous avons réussi à créer un jeu vidéo qui correspond au concept que nous avons développé et qui comporte les principales fonctionnalités auxquelles nous avons pensé. Avec plus de temps, l'objectif serait d'améliorer le design et de rajouter des niveaux. On pourrait aussi s'intéresser par exemple à des puzzles et énigmes plus complexes pour le joueur, ou encore à de nouvelles commandes, liées à l'ajout d'ennemis, ou d'autres personnages dans le monde, pour le peupler au fur et à mesure.

4.2 Bilans individuels

Chloé Boninn :

J'ai trouvé le sujet de ce projet très intéressant et motivant. De plus, m'a permis de me familiariser avec le langage C, notamment avec l'utilisation des pointeurs et la gestion de la mémoire. L'implémentation de A* et de la structure de données des files prioritaires m'a aussi permis de mieux voir l'importance du choix de la structure et de son implémentation. J'ai en effet rencontré plusieurs problèmes dus aux choix que j'avais fait et une implémentation différente m'aurait permis d'avoir un code plus clair et de gagner du temps.

D'un point de vue de la gestion de projet, contrairement à mon premier projet du premier semestre, les différentes parties étaient beaucoup moins indépendantes et il a fallu mieux s'organiser au niveau de l'utilisation du git, de la communication et du travail d'équipe, ce qui était très enrichissant.

Zoé Vernicos :

Ce projet a été très intéressant pour de différentes raisons : contrairement au premier ppii où l'on pouvait toutes coder nos parties séparément et l'assemblage était plutôt facile entre les différents morceaux, ici toutes les parties étaient plus ou moins liées et on a du s'adapter et travailler très souvent ensemble pour comprendre les différentes briques de codes qui ont été réalisées et comment les assembler. Ensuite le projet m'a permis de voir différentes manières d'utiliser le langage C ainsi que la librairie SDL. Enfin ce projet m'a permis d'avoir une meilleure idée de tous le travail nécessaire pour la réalisation d'un jeu, même si celui-ci à l'air "simple".

Lise Manière :

Dans ce projet a demandé plus de travail de groupe nous permettant ainsi de développer la communication et les compromis. J'ai trouvé intéressant de travailler avec quelqu'un et d'imaginer les possibles solutions à nos problèmes ensemble. De plus le projet étant en C, cela m'a permis de mieux m'approprier ce langage (en particulier les pointeurs).

Sylvie Sidler :

Le sujet de ce 2ème PPII a été très intéressant, faire un jeu vidéo était ludique et agréable à réaliser. Pouvoir travailler avec la même équipe qu'au précédent projet a rendu le travail de groupe plus simple à mettre en œuvre. Bien qu'il était aisé pour ma partie de travailler seule, la mise en commun a demandé plus de travail de groupe et d'organisation. La prise en main de la librairie SDL a été compliquée. J'ai eu l'occasion de toucher à chaque partie du jeu lors de l'assemblage du code ce qui a été enrichissant et me motive à améliorer ce projet dans le futur.

Olivia Aing :

Le projet proposé était intéressant, et m'a permis d'appliquer des concepts quant à la programmation de RPG que j'ai développé lors de mes créations personnelles avec RPG Maker (comme par exemple la variable histoire). Comme toujours, travailler avec cette équipe a été un véritable plaisir, avec des séances de brainstorming laissant un peu plus libre cours à notre imagination, et des séances de travail pour la mise en commun et la création de la démo complète qui étaient aussi fastidieuses que ponctuées de différentes exclamations (fussent-elles dirigées vers des problèmes miraculeusement réglés, ou au contraire de nouveaux bugs incompréhensibles). L'utilisation de git, et la gestion de conflits de fichiers a été bien plus importante que pour le projet précédent. Je suis aussi reconnaissante à mes équipières de leur compréhension, vis à vis de ma situation. Ce projet a été également une bonne occasion d'apprendre à utiliser des outils de programmation qui nous sont nouveaux, de manière autonome, ce qui est en quelque sorte ce qui sera attendu de notre part lorsque nous serons dans le monde du travail : une adaptation aux contraintes imposées, des recherches et des discussions avec nos pairs pour maîtriser ces nouvelles librairies (ou nouveaux langages). Je regrette un peu le fait que la documentation de la SDL ne soit pas vraiment claire, et présentait donc un cap supplémentaire à franchir, qui nous a pris beaucoup de temps à franchir, notamment au début.

5 Annexes

5.1 Sources

Sources pour l'algorithme A^* :

- <https://www.techno-science.net/definition/6469.html>
- <https://www.geeksforgeeks.org/a-search-algorithm/>

Apprendre la SDL en C :

- <https://zestedesavoir.com/tutoriels/1014/utiliser-la-sdl-en-langage-c/>

5.2 Preuves

Preuve de l'incrémentation correcte de la variable histoire :

Proposition :

Si on a répondu précédemment positivement, et qu'on est donc à la variable $2n$:

- si on répond positivement, il faut ajouter 2
- si on répond négativement, il faut ajouter 1

Si on a répondu précédemment négativement, et qu'on est donc à la variable $2n-1$:

- si on répond positivement, il faut ajouter 3
- si on répond négativement, il faut ajouter 2

Et on obtient les événements $2n+1$ et $2n+2$ qui sont les suivants.

Initialisation

Pour $n = 1$

$0 = 2k + 1$ avec $k = 0$

Réponse positive : $1 + 3 = 4 = 2m$ avec $m = 2$

Réponse négative : $1 + 2 = 3 = 2l + 1$ avec $l = 1$

On obtient l'événement 4 pour une réponse positive, et l'événement 3 pour une réponse négative.

Pour $n = 2$

$2 = 2l + 1$ avec $l = 1$

Réponse positive : $2 + 2 = 4 = 2p$ avec $p = 2$

Réponse négative : $2 + 1 = 3 = 2l + 1$ avec $l = 1$

On obtient aussi l'événement 4 pour une réponse positive, et l'événement 3 pour une réponse négative.

L'initialisation est donc vérifiée.

Hérédité

Supposons que la proposition soit vraie pour $2n$, et $2n-1$ avec $n \in \mathbb{N}$, et vérifions qu'elle le soit aussi pour l'incrémentation suivante.

Soit $n \in \mathbb{N}$.

— Cas $2n$

Réponse positive : $2n + 2 = 2m$ avec $m = (n + 1)$

Réponse négative : $2n + 1 = 2(n + 1) - 1 = 2m - 1$

On obtient l'événement $2m$ pour une réponse positive, et l'événement $2m - 1$ pour une réponse négative.

— Cas $2n - 1$

Réponse positive : $2n - 1 + 3 = 2n + 2 = 2m$

Réponse négative : $2n - 1 + 2 = 2n + 1 = 2(n + 1) - 1 = 2m - 1$

On obtient aussi l'événement $2m$ pour une réponse positive, et l'événement $2m - 1$ pour une réponse négative.

L'hérédité est donc vérifiée.

Conclusion

Ainsi, pour un événement histoire donné, la variable histoire sera toujours de la forme $2n - 1$ si on a répondu négativement, et $2n$ si on a répondu positivement, avec $n \in \mathbb{N}$

Cas particulier de 0 :

Pour $n = 0$

$0 = 2k$ avec $k = 0$

Réponse positive : $0 + 2 = 2 = 2l$ avec $l = 1$

Réponse négative : $0 + 1 = 2k + 1$ avec $k = 0$

On obtient l'événement 1 si la réponse est négative, et 2 si l'événement est positive, en utilisant les mêmes formules. On n'a donc pas besoin de traiter le cas de 0 à part.

5.3 Game Design Document

\debug_mode

Game Design Document

Introduction

- Résumé du jeu
- Inspiration
- Expérience joueur
- Plateforme
- Genre
- Audience visée

Concept

- Gameplay général
- Mécaniques primaires
- Mécaniques secondaires

Graphismes

- Palettes de couleurs
- Design

Audio

- Musique
- Effets sonores

Expérience de jeu

- Interface utilisateur
- Contrôles

Timeline de développement

Introduction

Résumé du jeu

Dans \debug_mode, on incarne une araignée nommée Kiki, à la recherche de ses souvenirs, et de ce qu'il s'est passé avec son monde. Pour cela, elle devra parcourir les différents niveaux, et fichiers, à la recherche de fragments à remettre au bon endroit pour débloquent le "jeu".

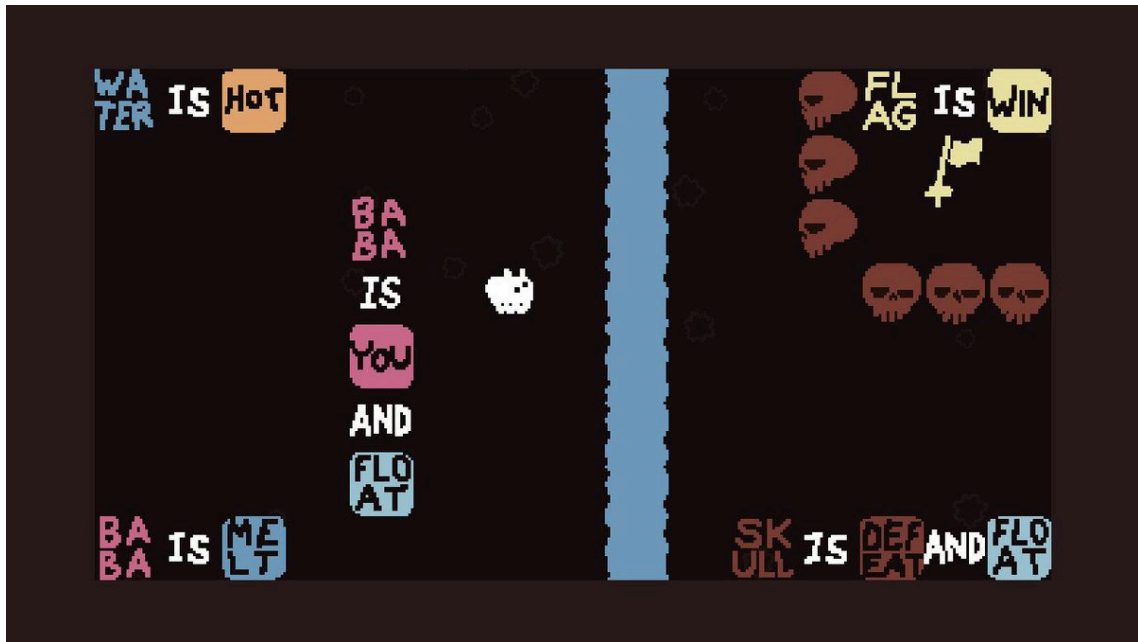
Inspiration

There is no game



Retrouver des morceaux de jeu
Créer / recréer le jeu à travers l'histoire

Baba is you



Vue 3e personne
Layout des maps
Jouer avec les règles
Puzzle, logique

Expérience joueur

Plateforme

Testé sur ordinateur (Linux, Windows)

Genre

RPG, Puzzle, Solo, Casual

Audience visée

Jeunes adultes,

Concept

Gameplay général

Des puzzles avec des interrupteurs, qui demandent à ce que l'on lance un `\debug_mode`, pour aller chercher certains éléments

Mécaniques primaires

Mécaniques nécessaires pour le jeu de base

Liste de commandes

<code>\debug_mode</code>	Lance le <code>debug_mode</code> , permet de passer à travers les murs
<code>\save</code>	Permet de sauvegarder sa partie
<code>\game_mode</code>	Permet de revenir à la version “normale” du jeu, avec les collisions

Mécaniques secondaires

Mécaniques additionnelles si on a terminé le jeu de base

<code>\freeze</code> et <code>\unfreeze</code>	Permet de geler et de dégeler le jeu Dans ce mode, impossible d'infliger des dégâts aux ennemis, mais leurs attaques en cours font encore des dégâts
<code>\god_mode</code> et <code>\vanilla_mode</code>	Permet de ne plus recevoir le moindre dégât
<code>\move_speed</code>	change la vitesse du personnage (1 = normal speed)

Graphismes

Palettes de couleurs

Minimaliste

4 palettes différentes :

niveau 0 -> NB

niveau 1 -> B

niveau 2 -> R

niveau 3 -> V

Design

Possibilité de faire un contraste

-> ingame : pixel art, level design qui ne prend pas toute la "fenêtre"

-> hors game : pixel art avec un peu plus de pixels, level design prenant toute la fenêtre, invite de commande indisponible

Audio

Musique


Effets sonores

Expérience de jeu

Interface utilisateur

Contrôles

Timeline de développement

 [TL] \debug_mode

5.4 Comptes rendus de réunions



Projet Pluridisciplinaire d'Informatique Intégrative 2
PPII 2
1A TELECOM Nancy - 2024

Compte-rendu de réunion

Mardi 5 mars 2024

1 Caractéristiques de la réunion

1.1 Informations générales

- Type : Réunion de chantier
- Lieu : Visioconférence Discord
- Début : 20 h 00
- Durée : 21 h 10
- Durée : 1 h 00

1.2 Membres présents

- Olivia AING (*Chef de Projet*)
- Zoé VERNICOS (*Secrétaire*)
- Chloé BONINN
- Lise MANIERE
- Sylvie SIDLER

1.3 Ordre du jour

- BRAINSTORMING

2 Échanges au cours de la réunion

Brainstorming puis choix du jeu

Durée : 1h00

Résumé du brainstorming :

- Idée que le jeu ressemble à un rpg
- Que ça ressemble aux premiers jeu zeldas (balade entre map et résolution de puzzle)
- intégration d'énigmes à résoudre
- jeu qui serait au début "incomplet" et qu'on débloquerait au fur et à mesure de l'avancée de l'histoire et du jeu
- But du jeu : résoudre des énigmes, trouver des items pour rendre le jeu de plus en plus complet
- rajout possible de mobs/ennemis plus tard (ce qui soulève des questions : déplacements mobs? pattern d'attaque? faire du tour par tour?, ajout d'une barre de vie dynamique?, mobs qui feraient des dégats au touché?)
- Pour sauvegarder le jeu : utilisation d'un tuple contenant l'identifiant de la map, les positions x et y dans la map, et une variable globale d'avancement de l'histoire (aussi pour éviter répétition des cut-scene au cas d'allers-retours du joueur)

Conclusion du brainstorming :

- Choix du type de jeu : jeu dont le but serait de le rendre de plus en plus complet grace à différentes actions, qvec des pouvoirs qui s'utiliseraient grâce à une sorte d'invit de commande
- Le jeu serait en vue de dessus
- Choix du nom du jeu :
`\debug_mode`

3 Prochaine réunion

Date prévue :

Jeudi 14 mars 2024

Ordre du jour (non exhaustif) :

- THÈME : WBS

TO-DO list :

Chloé BONINN	- Envoi du mail pour validation du sujet
Tout le monde	- Réfléchir au WBS et à un sujet de rechange en cas de refus

FIN DE LA RÉUNION



Projet Pluridisciplinaire d'Informatique Intégrative 2
PPII 2
1A TELECOM Nancy - 2024

Compte-rendu de réunion

Jeudi 14 mars 2024

1 Caractéristiques de la réunion

1.1 Informations générales

- Type : Réunion de chantier
- Lieu : TELECOM Nancy, cafétéria
- Début : 14 h 10
- Durée : 15 h 20

1.2 Membres présents

- Olivia AING (*Chef de Projet*)
- Chloé BONINN (*Secrétaire*)
- Lise MANIÈRE
- Sylvie SIDLER
- Zoé VERNICOS

1.3 Ordre du jour

- GESTION DE PROJET : Répartition des tâches
- Brainstorming

1.4 Backlog

Tout le monde	- Validation du projet par les professeurs
---------------	--

2 Échanges au cours de la réunion

Répartition des tâches

- Olivia : ouverture "terminal" (mettre le jeu en pause, avoir un overlay), dialogues, boîtes de dialogue (bipbip à chaque caractère), gestion de projet, faire une liste des pouvoirs pour le personnage.
- Chloé : fonctions pour les différents pouvoirs (variables pouvant prendre 3 états, ON, OFF et UNAVAILABLE, affichage carte de donjon, sauvegarde ($\text{tuple}(\text{variable histoire}, \text{map}_i d, \text{position}_x, \text{position}_y)$))
- Lise : carte -> murs, interrupteurs, portes, objets clefs, taille de la carte
- Sylvie : menu start, sfx
- Zoé : déplacement joueur, ouverture "terminal" (mettre le jeu en pause, avoir un overlay), Makefile

Tâches à partager : histoire, guide du jeu, musique, design

Tâches à voir dans un second temps : barre de vie et monstres, options de personnalisation (couleur de kiki, mixeur de volume, accessoires ?)

Brainstorming

- Kiki sera bleue. Au niveau du sprite : saute au lieu de marcher ? Il faudrait faire une vue de 3/4. Si trop compliqué de dessiner une araignée on fait un personnage avec des caractéristiques d'araignée ? - Utilisation d'un algorithme complexe : A* avec une carte ?, les ennemis ?, sur un boss pied avec de l'ombre ? -> la carte serait un pouvoir - Créer un menu joueur en plus du terminal. - Créer toutes les cartes avec différentes couleurs. Organiser sous forme de chapitre. Transition entre 2 cartes : s'assombrit puis affiche la nouvelle carte. Découper les cartes en carreaux/ cases. - Il faut gérer les sauvegardes. - Choix des touches pour le déplacement : les flèches ou Z,Q,S,D (les 2 doivent marcher). Espace pour attaquer. - Pour les tests, créer des petits programmes indépendants. - Mettre des regex dans le Makefile ? - Lancer le debug mode change une variable que Lise pourra utiliser pour déterminer si le mur peut être traversé par le personnage ou pas. - Les pouvoirs ont des contre parties : par exemple, debug mode permet de traverser les murs mais empêche d'interagir avec tous les éléments (interrupteurs...), god mode permet d'être invincible mais on ne peut pas attaquer. - S'occuper de l'affichage complet, de la variable d'histoire... - Au niveau de l'organisation git on crée chacune une branche et on merge quand on est sûr que ça marche.

- Un personnage tutoriel en forme de livre expliquera les règles, rajouter la possibilité de choisir de faire ou non le tutoriel. Rajouter une variable d'amitié avec le tutoriel (Se vexe si on le skip) ? Rajouter une aide de jeu si le joueur est bloqué ?

- Possibilité d'un boss "pied" / "chaussure".

- Le jeu sera dans un premier temps noir et blanc et on découvre les couleurs une par une. Au niveau design, il y aura des objets à découvrir (par exemple : les couleurs), des coffres (mettre des armes (ducky-roquette, chewing-gum) ? aléatoire ?), des portes et des interrupteurs, des portes avec des clés ?

- On fait un premier arc de l'histoire qui pourrait continuer ensuite.

- Le jeu ne doit pas être trop compliqué. Pour créer et organiser le code : faire des chapitres.

- Si on a le temps faire un guide de jeu (présentation des personnages...)

- Un personnage mur qui bouge, il aurait une couleur différente comme ça visible qu'après certains chapitres.

Tâches supplémentaires :

- Il faudra coder les sounds effects, Olivia mettra des places holder en attendant.
- Rajouter un mixeur de volumes.
- Rajouter possiblement le choix de la couleur de Kiki.
- On améliorera le design après avoir codé le jeu.
- Création d'un inventaire (nombre de places limitées ?, s'affiche dans le menu joueur, avoir une arme de base qu'on peut modifier)
- Rajouter l'utilisation de la souris ? (pointer sur une carte pour voir le chemin le plus court jusqu'au point désigné ?)
- Barre de vie et monstres. Créer un god mode.

3 Prochaine réunion

Date prévue :

Jeudi 21 mars 2024

Ordre du jour (non exhaustif) :

- réunion d'avancement

TO-DO list :

Tout le monde	- recherches - commencer la programmation
---------------	--

FIN DE LA RÉUNION



Projet Pluridisciplinaire d'Informatique Intégrative 2
PPII 2
1A TELECOM Nancy - 2024

Compte-rendu de réunion

Jeudi 4 avril 2024

1 Caractéristiques de la réunion

1.1 Informations générales

- Type : Réunion de conception-avancement
- Lieu : TELECOM Nancy, salle Médialab
- Début : 13 h 00
- Durée : 1 h 00

1.2 Membres présents

- Olivia AING (*Chef de Projet*)
- Sylvie SIDLER (*Secrétaire*)
- Chloé BONINN
- Lise MANIÈRE
- Zoé VERNICOS

1.3 Ordre du jour

- PRÉCISION DES OBJECTIFS : Revoir les objectifs de manière plus précise pour être sûres de connaître les tâches de chacune
- AVANCEMENT : Point sur l'avancement de chacune

1.4 Backlog

Olivia AING	- Ouverture de la fenêtre de terminal et la fenêtre de dialogue	En cours : mais problème d'inversion des 2 (cf. partie Avancement)
	- Dialogues, liste de pouvoirs pour le personnage et gestion de projet	En cours / à réaliser
Chloé BONINN	- Plan du donjon	En cours / recherches
	- Fonctions pour les différents pouvoirs 2	Commentaire (Terminé ou non, blocages ?)
Sylvie SIDLER	- Menu de démarrage	En cours
Lise MANIÈRE	- Carte de la salle actuelle	En cours / recherches
Zoé VERNICOS	- Makefile	Terminé
	- Déplacement joueur	En cours

2 Échanges au cours de la réunion

PRÉCISION DES OBJECTIFS : Revoir les objectifs de manière plus précise

Ordre des tâches pour la carte totale de donjon

Carte entièrement visible → A* → découverte de la carte selon l'avancement du joueur → pouvoirs

Carte visible : affichage de la carte complète du donjon en quadrillage de toutes les salles

Algorithme A* : il faudra utiliser comme données en entrée la carte créée par Chloé et gardée en sauvegarde. Pendant l'exécution de l'algorithme, il faudra pouvoir suivre la position du joueur sur la carte et la mettre à jour selon ses déplacements.

Découverte des salles sur la carte : après avoir visité une pièce, la case sur le plan passe de assombrie à visible en noir/blanc ou en couleurs.

Pouvoirs : après saisie du pouvoir dans le terminal une fonction prendra en paramètre la string correspondant à l'input et appelle la fonction du pouvoir correspondant.

Niveau 1 : juste 1 pièce sans murs visibles, le joueur a donc besoin de l'algorithme A* pour trouver son chemin. Peut faire partie du tutoriel pour introduire l'accès à l'invite de commandes.

Menu de démarrage

Proposer un menu options avec pour l'instant uniquement une page vide sauf la flèche de retour. Elle pourra être utilisée pour la personnalisation de Kiki, changement des commandes, ...

Définir une structure Save pour unifier la manière d'enregistrer les sauvegardes des éléments suivants :

- var d'avancement de l'histoire : en int
- identifiant de la carte (quelle salle) : en int
- la position sur la carte en x et y : en 1 tuple ou 2 int
- quelles salles ont été visités (pour l'avancement de la carte) : en liste

Réflexion sur le plan

Il faudra peut-être créer un objet mur pour les cases du plan. Le format pour la fenêtre du jeu et chaque case du plan sera à définir dans une prochaine réunion.

Pour plus tard, on pourra faire des pixelArt pour les murs et sols. Pour l'instant, on peut utiliser un code couleur comme :

- le sol en noir
- les objets interactifs avec Kiki ("bumpable") comme les murs en blanc

Attention à la bordure de fenêtre de jeu qui n'est pas accessible pour Kiki. Il faudra mettre une différence de couleur entre le cadre et le jeu

Astuce de debug

Pour debug on peut utiliser la commande :

`gdb nomFichier run`

AVANCEMENT : Point sur l'avancement de chacune

Chloé :

S'est renseignée et a l'algorithme du A*

Sylvie :

A résolu ses problèmes de SDL.

Le menu s'affiche avec 2 options pour commencer une nouvelle partie ou charger la dernière sauvegarde. Une page s'affiche le temps de charger le jeu en fonction du choix précédent.

Lise :

A résolu ses problèmes de SDL et commence les tests de cartes.

Zoé :

Le Makefile est opérationnel

Le problème de la fonction IMG_Load non trouvée par SDL est résolu, mais maintenant la fonction ne trouve pas le fichier image à ouvrir.

Olivia :

Les 2 rectangles correspondant au terminal et à la fenêtre de dialogue s'affichent correctement au dessus de la fenêtre de jeu. Problème sur l'affichage de la fenêtre de dialogue et la fenêtre de terminal inversé par rapport aux touches clavier prévues.

3 Prochaine réunion

Date prévue :

Jeudi 11 avril 2024

Ordre du jour (non exhaustif) :

- AVANCEMENT : Point sur l'avancement de chacune
- FÊNETRE : Définir les dimensions de la fenêtre de jeu dans la fenêtre de l'application

TO-DO list :

Chloé BONINN	- Définir les fonction des pouvoirs
Sylvie SIDLER	- Définir la structure de sauvegarde de partie
Lise MANIÈRE	- Tests de cartes
Zoé VERNICOS	- résoudre le problème de IMG Load qui ne trouve pas limage
Olivia AING	- Résoudre le problème de l'affichage inversé des fenêtres de dialogue et de terminal

FIN DE LA RÉUNION