

Aldwin Baïetto
Lucie Cayment



Rapport du projet de programmation avancée

Implémentation d'un jeu type roguelike

ROGUEPUNK 2021

avril/mai 2021

Sommaire

Introduction	2
Choix d'implémentation	2
L'univers du jeu	2
Les principales classes	4
Les classes à héritage	5
La classe Actor	5
La classe Tresor	5
La classe Objet	6
La classe CityMap	7
Les classes sans héritage	7
La classe Game	7
La classe SystemeDeCommande	8
Manuel d'utilisation	9
Contrôles	9
Séquences de jeu	11
Gestion du projet	12
Répartition des tâches et gestion du temps	12
Les tests	13
Problèmes rencontrés et améliorations possibles	14
Problèmes	14
Améliorations	14
Conclusion et retour d'expérience	15

1.Introduction

a. Choix d'implémentation

Comme proposé sur l'énoncé, un tutoriel pour implémenter un jeu type roguelike pouvait être suivi. Ce tutoriel utilise les librairies Roguesharp et RLNET qui sont deux librairies spécialisées pour la création de jeux roguelike. Après un court aperçu des possibilités que ces librairies nous offraient et à cause de la contrainte de temps, nous avons choisi de suivre entièrement ce tutoriel.

Cependant nous avons souhaité en amont imaginer les classes que nous pourrions utiliser ainsi que les variables les plus importantes. La plupart des classes imaginées sont dans le tutoriel.

Il était important d'ajouter une plus value au jeu en addition au tutoriel. En effet, ce dernier est loin d'explorer toutes les caractéristiques d'un jeu roguelike. Nous avons ainsi axé notre travail sur la gestion des ressources ainsi que sur le combat. Ce dernier se présente comme un combat avec actions à choix. Quatre actions possibles pour le joueur : attaquer, défendre, se soigner ou fuir.

Des précisions sur l'implémentation sont développées dans la partie 2 de ce rapport.

b. L'univers du jeu

Les jeux roguelike se déroulant souvent dans un environnement heroic-fantasy il nous semblait intéressant de sortir de cette thématique et d'en choisir une autre. Nous avons choisi l'environnement cybernétique en référence au jeu sorti plus tôt dans l'année : Cyberpunk 2077.

L'univers cybernétique se définit par des objets technologiques ou des prothèses cybernétiques intelligentes. Le joueur incarne donc un cyborg en quête d'aventure et qui s'attaque à la tour de Roguecity. Son but est de gravir tous les étages de la tour en terrassant les différents ennemis et obstacles.


Le jeu débute par un écran d'accueil présentant l'équipement que le joueur va pouvoir choisir. Cet équipement correspond au type de jeu de joueur. En effet, si il choisit l'équipement dit "furtif" (des jambes robotisées) cela lui permet de fuir les ennemis et d'éviter le combat. Il peut aussi choisir l'équipement offensif (les bras

armés) pour attaquer avec un taux de réussite de 70% (contre 50% sans l'équipement). Enfin, avec l'équipement "défensif" (le blindage), aucune défense n'échoue.

Différents objets sont répartis de manière aléatoire dans chaque niveau (qui est composé d'un nombre aléatoire de salles). Ces objets sont ramassés lorsque le joueur marche dessus. On peut trouver des bonus d'attaque et de défense qui rapportent chacun un point d'attaque ou de défense. Ces objets sont représentés

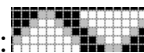
par les icônes suivantes :  et .

Le joueur peut également trouver des kits de réparations pour son équipement qui sont stockés dans un inventaire et utilisables à tout moment. Ces


kits rapportent 20 points de vie au joueur. Ils sont représentés par le symbole .


Un des objets les plus importants est la carte d'accès au téléporteur. En effet, ce téléporteur est la seule voie d'accès à l'étage supérieur. Sans cette carte, le téléporteur reste verrouillé. La clé est disposée dans une des pièces du niveau (mais jamais dans les salles où se trouvent les téléporteurs (vers l'étage inférieur ou

supérieur)). Son symbole est le suivant : .

Les téléporteurs sont représentés par des flèches pointant vers le bas pour aller vers l'étage inférieur et vers le haut pour aller vers l'étage supérieur : . Il faut cependant noter qu'il n'est pas possible de redescendre de niveau. Le téléporteur vers l'étage inférieur est donc inutilisable.

Enfin les ennemis présents dans les niveaux sont des droïdes et des robots soldats. Ces ennemis ont chacun des caractéristiques différentes (points d'attaque, de défense, points de vie, etc). Le droïde est un ennemi de type défensif et il est

représenté par le symbole . Le robot soldat est de type offensif et est

représenté par le symbole .

Une fois éliminés, ces ennemis donnent au héros des crédits (équivalent de gold) et des points d'expérience.

2. Les principales classes

Le projet se compose de 32 classes dont 6 interfaces. Toutes ces classes sont rangées dans des dossiers propres à leurs caractéristiques. On trouve 7 dossiers :

- BigColors : contient les fichiers propres aux couleurs utilisées (*Palette* et *Color*). Les palettes de couleurs ont été sélectionnées pour convenir au thème cybernétique.
- Core : contient les fichiers liés aux éléments graphiques et à leurs actions et interactions entre eux (*Actor*, *Enemy*, *Player*, *CityMap*, *Door*, *Objet*, *Tresor*, *TresorRamassable*, *Soin*, *Teleporteur*)
- Enemies : regroupe les classes *SoldierRobot* et *Droid* qui sont deux classes-filles de *Enemy* ainsi que la classe *StandardMove* qui gère le mouvement des ennemies.
- Interfaces : regroupe des 6 interfaces (*IActor*, *IBehavior*, *ISpeed*, *IDessin*, *IObjet*, *ITresor*)
- Objets : contient uniquement la classe *KitDeReparation*, objet utilisable par le joueur.
- Systems : contient les fichiers liés aux fonctions génératrices d'éléments (la map, les objets et les trésors (*ItemGenerator*, *MapGenerator*, *TresorGenerator*, *Pool*)) ainsi que la gestion de la vitesse (*SpeedSystem*) et des messages (*MessageInit*, *MessageLog*) pour le joueur. Les entrées utilisateurs sont également gérées via la classe *SystemeDeCommande*.
- Tresor : contient les différentes classes qui correspondent aux trésors ramassables, à savoir les renforcements pour l'armure (*RenfoArmure*), pour l'attaque (*RenfoAttaque*), ainsi que la carte d'accès (*CarteMag*) pour le téléporteur.

Les prochains paragraphes visent à présenter les principales classes implémentées dans ce projet, à défaut de présenter toutes les classes (ce qui ne semblaient pas pertinent au vu de leur grand nombre). Ces principales classes se découpent en deux types : les classes à héritage et les classes sans héritage.

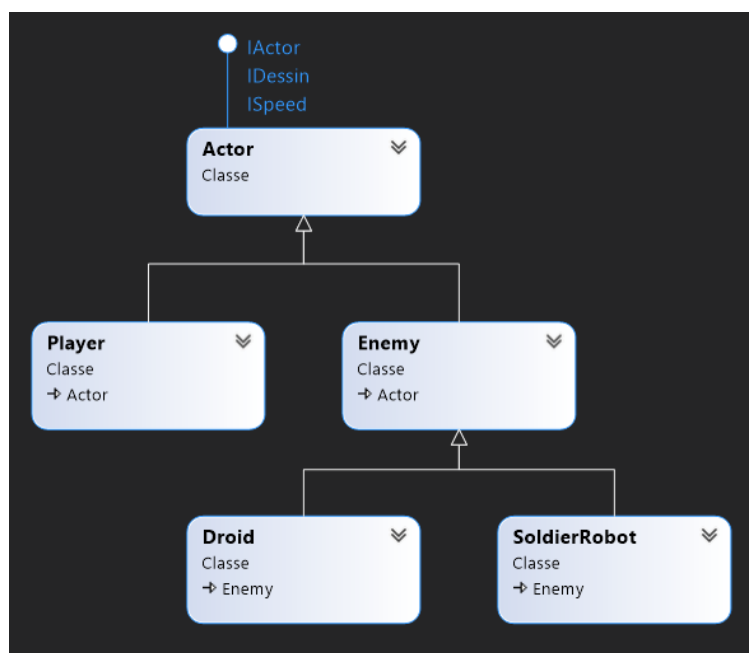
a. Les classes à héritage

Parmi toutes ces classes, quatre classes-mères ont été créées : *Actor*, *Objet*, *Enemy* et *CityMap*.

La classe Actor

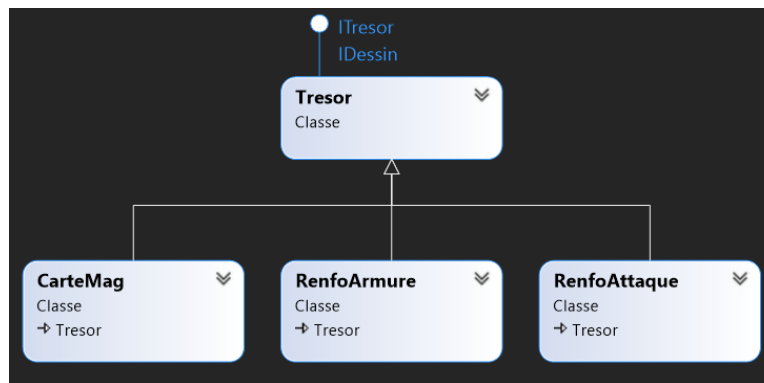
La classe *Actor* est composée de 17 champs qui représentent toutes les caractéristiques d'un acteur. Un acteur est défini soit comme un joueur soit comme un ennemi. Ainsi la classe *Player* et la classe *Enemy* héritent toutes les deux de la classe *Actor*. Il est important de noter que la classe *Actor* hérite elle-même des trois interfaces : *IActor*, *IDessin* et *ISpeed*.

Il existe deux types d'ennemis : les droïdes et les robots soldats. Ces deux classes héritent naturellement de la classe *Enemy*, comme le montre la figure suivante.



La classe Tresor

La classe *Tresor* est une classe qui définit une méthode commune aux trésors répartis sur la map : la méthode pour ramasser ces trésors. Cette méthode *PickUp* définit ainsi les changements de statistiques en fonction du trésor ramassé. Il en existe trois : la carte magnétique d'accès au téléporteur, le renforcement pour l'armure et le renforcement d'attaque. La figure suivante résume les relations d'héritages entre ces différentes classes.



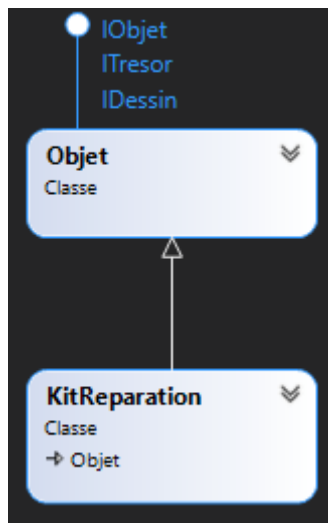
A noter que, tout comme la classe *Actor*, la classe *Tresor* hérite des interfaces *ITresor* et *IDessin*.

Lorsque le renforcement d'armure ou d'attaque est ramassé par le joueur, cela fait gagner un point dans la catégorie correspondante. Lorsque la carte magnétique est ramassée, cela donne l'accès au joueur au téléporteur pour l'étage suivant. Sans cette carte, l'accès au niveau supérieur est impossible.

La classe *Objet*

La fonction de la classe *Objet* est assez proche de celle de la classe *Tresor* car toutes les instances de ces classes peuvent être ramassées mais seules les instances de la classe *Objet* peuvent être utilisées par le joueur. Dans notre cas la seule classe-fille est le kit de réparation qui permet au joueur de regagner 20 points de vie.

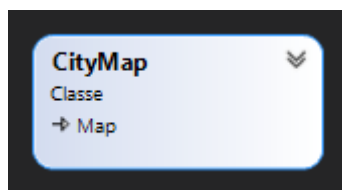
La classe-mère hérite également de trois interfaces : *IObjet*, *ITresor* et *IDessin*. Ce kit de réparation peut être utilisé durant le jeu ou durant un combat. Le joueur commence la partie avec un kit de réparation dans son inventaire.



La classe CityMap

La classe **CityMap** provient en partie du tutoriel présenté en début de rapport. Cette classe est la classe-fille de la classe **Map.cs** qui est comprise dans la librairie **Roguesharp**. Cette classe **Map** contient tous les champs et méthodes pour créer une map.

Dans notre cas, la classe **CityMap** a pour mission de gérer les portes (ouverture notamment) ainsi que l'ajout du joueur, des ennemis, des trésors et des objets. On y trouve également des méthodes de dessin (pour la map en elle-même et pour les statistiques du joueur et des ennemis) et encore des méthodes utilitaires.



b. Les classes sans héritage

La classe Game

La classe **Game** est la classe principale du programme. Elle contient la déclaration de toutes les variables globales (taille des consoles et sous-console notamment) ainsi que le main et les méthodes liées à la mise à jour de la console.

C'est dans cette classe que sont instanciés les objets d'affichage sur l'écran ou dans la sous-console de message pour le joueur (respectivement **MessageInit** et **MessageLog**).

Les fonctions d'initialisation ont été regroupées dans deux méthodes distinctes : *CreateMap* et *Init*. *CreateMap* permet de générer une carte et *Init* contient les instanciations des objets du scheduler (gestion du déplacement et de la vitesse), du système de commande et du joueur. Il s'y trouve également la lecture d'un fichier texte qui présente les règles du jeu.

Les méthodes de mise à jour sont les suivantes :

- OnRootConsoleUpdate : ici sont gérées toutes les entrées utilisateurs en fonction des situations (lorsque le joueur a choisi son équipement ou non, lorsque le joueur est en combat ou non, etc)
- OnRootConsoleRender : ici les consoles sont remises à jour quand c'est nécessaire (par exemple le joueur ou les ennemis se sont déplacés, les statistiques ont évolué, etc).

Cette classe Game est le point de départ de toutes les méthodes et classes.

La classe SystemeDeCommande

En suivant la classe Game, la classe SystemeDeCommande est celle qui est directement appelée lors d'une entrée utilisateur.

Il existe différentes situations où l'entrée utilisateur est nécessaire :

- lors des combats : la méthode *ChoiceCombat* est alors appelée
- pour l'affichage des règles du jeu/menu : la méthode *DisplayMenu* est appelée
- pour le choix de l'équipement : la méthode *ChoixClasse* est appelée
- pour tout déplacement du joueur : la méthode *MovePlayer* est appelée
- pour l'utilisation du kit de réparation : la méthode *UtiliserKit* est appelée

Lorsque la partie ou les combats sont finis, des méthodes sont également appelées pour gérer les actions liées (*EndTurnCombat*, *EndPlayerTurn*, *EndCombat*).

Le mouvement des ennemis est géré par deux méthodes (*MoveEnemy* et *ActivateEnemy*). Ces méthodes gèrent respectivement le mouvement de l'ennemi et lorsqu'il est actif (donc que le joueur est dans son field of view).

Un visuel de combat assez simpliste est créé dans la méthode *LaunchCombat*. Ce visuel pourrait être amélioré avec des graphismes plus développés pour rendre l'expérience de jeu plus agréable.

Enfin, et toujours en restant dans l'approche combat, il existe des méthodes pour l'attaque et la défense des acteurs (donc le joueur et les ennemis) ainsi que pour leur mort (lorsque leurs points de vie atteint 0).




La dernière méthode est la méthode qui gère la montée en niveau du joueur qui occure lorsqu'il atteint 25 points d'expérience.

3. Manuel d'utilisation





a. Contrôles


Afin de se mouvoir et de traverser la tour de Roguepunk 2021, le joueur a plusieurs options et dispose de deux catégories de touches différentes.




Avant même le début de l'exploration de la tour. Un menu expliquant le contexte et le but du jeu apparaît. Le joueur va pouvoir sélectionner sa classe parmi les trois disponibles.

- La classe "offensif" sélectionnable avec la touche 'O' . Cette classe va octroyer des dégâts supplémentaires au joueur et augmenter ses chances de porter un coup à l'ennemi lors d'un combat. Il aura également une faible chance d'asséner un coup critique lors d'un combat, infligeant le double de dégâts ordinaires.
- La classe "défensif" sélectionnable avec la touche 'D' . Le joueur ayant choisi la classe défensive se verra attribuer des points de défense supplémentaires et pourra parer à coup sûr l'attaque de son ennemi.
- La classe "Furtif" sélectionnable avec la touche 'F'  est la dernière. Un joueur furtif verra sa statistique de vitesse augmenter, il pourra ainsi se déplacer plus rapidement et sera toujours capable de fuir un combat du premier coup.





Une fois sa classe choisie, le joueur sera transporté dans la tour et pourra se déplacer. Pour cela, nous avons choisi les touches directionnelles du clavier. Le joueur peut aller dans quatre directions présentes sur ces touches :


- La touche “haut”  qui permet d’aller vers le haut.
- La touche “droite”  , permet de se déplacer vers la droite de l’écran.
- La touche “gauche”  , permet de se déplacer vers la gauche.
- La touche “bas”  , permet au joueur d’aller vers le bas.


En plus de ces quatre touches, une cinquième va permettre de naviguer dans la tour, la touche “/”  , va permettre au joueur, si il le peut, d’emprunter le téléporteur vers le niveau supérieur de la tour.

Ensuite, nous avons les touches qui permettent de réaliser différentes actions lors du jeu. Lorsque le personnage du jeu n’est pas en combat contre un ennemi, le joueur a la possibilité d’ouvrir un menu lui réexpliquant ces objectifs ainsi que les différents contrôles du jeu. La touche ‘M’  permet d’ouvrir ce menu. Pour le quitter, il suffit d’appuyer sur la touche ‘X’  . Hors combat, le joueur a aussi la possibilité de se soigner en utilisant, si il en a un, un kit de réparation. Pour cela, il doit appuyer sur ‘P’  .

Pour finir, nous avons les contrôles de combat. En effet, quand le joueur et un ennemi entrent en contact, le mode combat est activé et les contrôles changent. A ce moment-là, le joueur ne peut plus se déplacer et dispose de quatre options.

- La touche ‘A’  , qui lui permet d’attaquer en prenant en compte ses chances d’attaque et ses dégâts.
- La touche ‘D’  , pour se défendre, et si elle réussit, bloquer le nombre de dégâts en fonction de ses points de défense.
- La touche ‘P’  , qui a la même fonction que hors combat. Néanmoins, l’utilisation d’une potion en combat lui fait perdre son tour.
- Enfin la touche ‘F’  , qui laisse au joueur 50% de chance de s’enfuir du combat, sauf si il a choisi la classe furtive.

Si jamais le joueur venait à mourir lors d’un combat, il a la possibilité de recommencer une partie de zéro en appuyant sur la touche ‘R’  .

Une dernière touche est accessible à tout moment par le joueur, c'est la touche "Echap" . Cette dernière touche permet simplement de quitter le jeu et ce, à n'importe quel moment de la partie. Néanmoins, toute la progression sera perdue.

b. Séquences de jeu

Toutes les sessions de jeu commencent de la même façon. Devant un menu explicatif, le joueur va pouvoir sélectionner sa classe avant de commencer à explorer la tour.

Une fois la classe choisie, le jeu peut commencer.

Comme vu précédemment, les salles de la tour sont parsemées de monstres et d'objets divers à ramasser. Les kits de réparation ont 15% de chances d'apparaître dans n'importe quelle salle de l'étage. Pour ce qui est du renforcement d'attaque, d'armure et de la carte magnétique, un seul élément de chaque se trouve par étage. De plus, seule la carte est nécessaire pour déverrouiller le téléporteur et pouvoir continuer de progresser dans le jeu. De plus, ces trois derniers ne peuvent ni se trouver dans la salle d'arrivée du joueur, ni dans la salle du téléporteur vers le niveau suivant. Enfin, un dernier élément peut apparaître dans les autres salles que celles des téléporteurs, les ennemis.

Entrer en contact avec un ennemi signifie entrer en phase "combat". Cette phase reprend le fonctionnement d'un système de combat au tour par tour. Avant chaque tour, le joueur va pouvoir sélectionner une action qui sera réalisée avant, ou après l'ennemi, en fonction de leur statistique de vitesse. Ce système de combat permet d'apporter une dimension stratégique plus poussée au jeu qu'un roguelike peut habituellement proposer. Le joueur doit plus faire attention à la disposition des salles et de ces objets, pour ne pas se retrouver coincé dans un combat avec peu de point de vie et sans la possibilité de fuir. De plus, pour éviter que le joueur passe son temps à fuir avec la classe furtive, au fil des étages, les ennemis accélèrent rendant difficile l'exploration sans combattre. Afin de récompenser le joueur qui se bat, un système d'expérience et de niveau a été mis en place. En effet, si le joueur tue assez d'ennemis il pourra augmenter de niveau et ainsi de statistiques pouvant plus facilement faire face aux ennemis qui augmentent leur statistiques au fur et à mesure des étages grimpés, rendant d'autant plus punitif l'esquive de combat au début.

Afin de gérer le déplacement des ennemis réactifs aux déplacements du joueur, le système de “scheduling” du tutoriel a été utilisé. En effet, chaque personnage (joueur et ennemis) dispose d’une vitesse. Quand le personnage est créé, il est ajouté au “schedule” et sa vitesse va déterminer le temps nécessaire qu’il devra attendre entre deux actions. Ainsi, moins l’indice de vitesse est grand, plus il ira vite. Ainsi le joueur qui commence avec une vitesse de 10 sera plus rapide qu’un droïde avec une vitesse de 14. Mais si le joueur n’augmente pas cette statistique, le droïde deviendra plus rapide car à chaque étage et la statistique de vitesse du droïde diminuera. Quand sa statistique sera à 9, il pourra agir plus souvent que le joueur et finira par le rattraper. Un ennemi mort se voit retirer du “schedule” et chaque étage le “schedule” se réinitialise afin d’éviter des bugs. Chaque personnage a une statistique limitée à 1, car à partir de 0, le déplacement n’aurait plus d’impact sur le “schedule” et le personnage en question serait le seul à pouvoir se déplacer.

Si le joueur meurt, l’écran de Game Over apparaît, remplaçant la carte. Seules deux options sont alors possibles, quitter le jeu ou relancer une partie. Plus on grimpe dans les étages, plus les salles rétrécissent, augmentant ainsi leur nombre, le nombre d’ennemis et le nombre de kits de soins. Rendant la tâche plus ardue.

Une fois arrivé au dernier étage, après en avoir traversé sept, le joueur remporte la victoire et arrive sur l’écran de victoire. Il peut alors relancer une partie ou bien quitter le jeu.

4. Gestion du projet

a. Répartition des tâches et gestion du temps

Ce projet a commencé le 22 mars 2021 et s’est terminé le 6 mai 2021. Afin de gérer les tâches nous avons créé un tableau avec les dates et les différents objectifs. Ces objectifs évoluent au fil du projet et sont attribués à chacun durant les réunions ponctuelles réalisées pendant le mois et demi de projet.

	Lucie						
	Aldwin						
Planning	22/03/2021	29/03/2021	05/04/2021	12/04/2021	19/04/2021	26/04/2021	03/05/2021
Prendre en main RogueSharp	X	X					
Faire un schéma UML des classes et des méthodes associées (nécessaires)	X	X					
Créer le projet		X					
Mise en place de la console et des sous-console (stats perso, stats monstres, map, messages)		X	X				
Construction de la map		X	X				
Finir le tutoriel Roguesharp		X	X	16/04/2021			
Maj a jour du schéma des classes			X				
Refixer objectifs pour après le tutoriel					23/04/2021		
Création d'un menu pour le choix des classes				X	X		
Entrée utilisateur				X			
Résoudre souci clavier QWERTY							X
Mise en place du mode combat avec la console log				X	X	X	X
Gestion du système de combat				X	X	X	X
Créer les différents niveaux et les changements d'un niveau à l'autre (clé + porte et téléporteur)						X	X
Simplifier le schedule du tuto et l'ajouter dans le code					X	X	
Création de zones safe (salle ou on spawn et salle teleporteur)					X		
Distribution aléatoire des ressources (attaque, défense, potions) dans les salles					X	X	
Ajouter une musique							
Ajouter l'option de potion dans le combat							X
Ajouter les capacités en plus suivant le stuff choisi							X
Faire la fin du jeu et pourquoi pas un moyen de restart une game							X
Ajout d'un système d'expérience et de niveaux (Adapté à la classe choisie)							X
Modifier les variables de PV (monstres et joueur) en fonction des niveaux							X
Ajouter des illustrations (changer le font file)							X
Commenter le code							X
Rapport							

A l'aide d'un code couleur, les tâches étaient répartis de manière équitable. Le tutoriel a été réalisé par chacun de nous afin de bien maîtriser le cœur du code. A partir du 16 avril, nous avons commencé à personnaliser le programme. Les principales tâches liées au combat et au choix des classes ont été réalisées par Lucie. Parallèlement Aldwin a travaillé sur la gestion des niveaux, des portes, des objets et des trésors. Les derniers jours du projet ont eu pour rôle de faire de légères améliorations et résolution de bug trouvés durant les tests. Ces améliorations et résolutions ont été partagées entre nous. Le rapport a été écrit par le binôme.

b. Les tests

Les tests ont pris la forme de séquences de jeu durant lesquelles ils étaient importants d'explorer toutes les fonctionnalités du jeu afin de déterminer celles qui n'étaient pas au point, manquantes ou déficientes. Dans ce type de projet, le jeu est la meilleure façon d'identifier les problèmes. Ces séquences de jeu ont été réalisées durant l'entièreté du projet. Cependant vers la fin les séquences sont devenues plus complètes et donc plus longues, grâce à l'avancement du projet.

5. Problèmes rencontrés et améliorations possibles

a. Problèmes

Grâce aux nombreux tests effectués tout au long du projet, peu de bugs sont encore présents dans la version finale du jeu.

Le principal problème encore présent est la présence par moment d'ennemi-mur. En effet, nous avons remarqué qu'à partir du deuxième étage qu'un ennemi ayant l'apparence d'un mur apparaît. Ce monstre a la particularité de toujours détecter le joueur et de toujours se déplacer vers lui, même s'il se trouve dans une salle non explorée. Ce bug peut totalement ruiner une partie à cause du comportement de l'ennemi. En effet, il a le comportement d'un ennemi normal et donc à son contact, on rentre en phase combat avec lui. Mais il n'attaque pas et ne prend pas de dégâts et ainsi ne meurt jamais. Toutefois, on peut fuir le combat, mais il suffit que deux de ses ennemis se trouvent au même étage et encerclent le joueur dans un couloir pour stopper toute avancée possible. Nous ne savons pas pourquoi ce bug se produit. Il se peut que ce soit un problème lors de la gestion de mort d'un ennemi et que les informations sont mal transmises lors du passage d'un étage à l'autre, ce qui expliquerait pourquoi aucun ennemi-mur n'apparaît au premier étage, mais aucune piste de correction du bug n'a été concrètement identifiée. De plus, la raison qui donne à l'ennemi l'apparence d'un mur est aussi inexpiquée.

D'autres bugs mineurs existent, mais ne viennent pas entacher l'expérience de jeu.

En effet, on peut voir sur les écrans du menu principal, du menu durant la partie et du menu de combat notre personnage tel qu'il est dans la tour. Bien qu'on ne puisse pas le déplacer, il est présent.

Il arrive aussi, à la mort du joueur, que l'écran de Game Over ne s'affiche pas.

b. Améliorations

Plusieurs pistes d'améliorations du jeu s'offrent à nous. En effet, outre la correction de ces bugs, d'autres options sont disponibles.

Nous pourrions augmenter le nombre d'ennemis et rajouter des boss avec diverses spécificités, comme des compétences ou des comportements différents (plutôt agressif, ou plutôt défensif par exemple).

Un système d'amélioration en jeu, par le biais d'un magasin a aussi été imaginé, d'où la présence d'une monnaie virtuelle, les crédits, qui servent pour le moment plus comme un score qu'une réelle fonction. Pour cela, des personnages non joueurs (PNJ) pourront être ajoutés.

Une dimension visuelle aussi est envisageable. Avoir des visuels des différentes classes choisies par exemple, ou une interface de combat plus à l'image des jeux au tour par tour, avec une vision du personnage de dos et en face l'ennemi en contact.

Enfin, un ajout d'une plus grande variété d'objets avec diverses utilités, comme des capacités à usage unique ou un système d'équipement pour le personnage.

D'autres apports minimes comme une musique, un scénario plus avancé et une meilleure rejouabilité sont également possible dans l'optique d'amélioration du jeu.

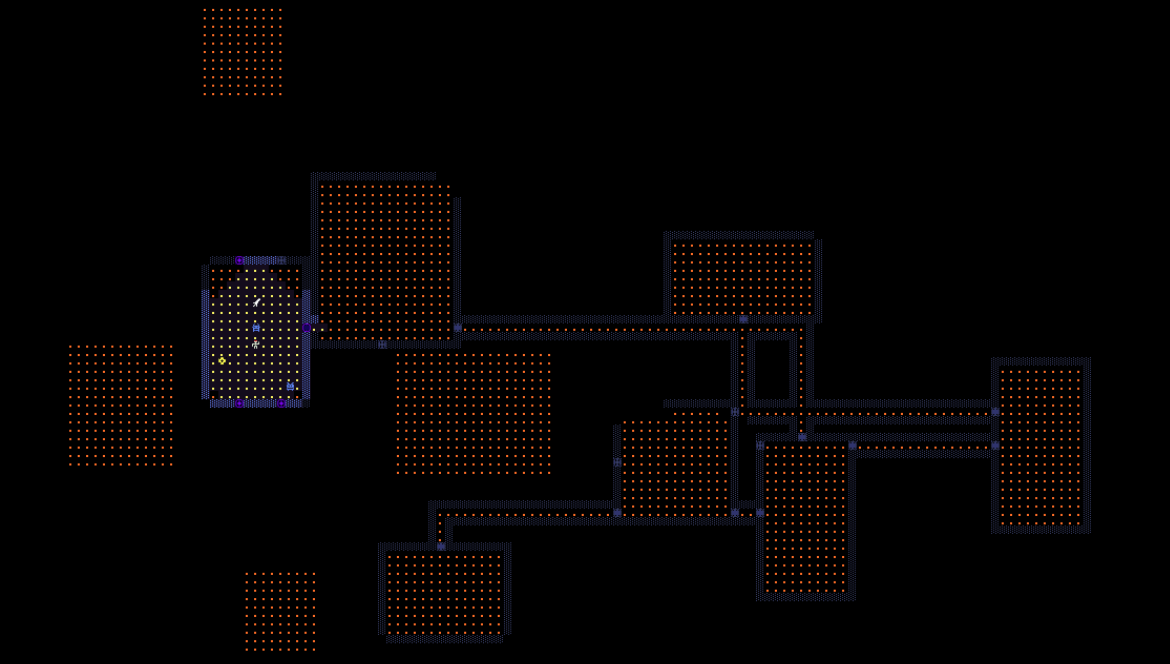
6. Conclusion et retour d'expérience

La conclusion de ce projet est assez positive. Les objectifs fixés ont été atteints et des améliorations ont été identifiées. L'utilisation de github nous a permis de travailler parallèlement sans pour autant nous gêner. La contrainte de temps était cependant dommage. En effet, en si peu de temps il est difficile d'utiliser tout le potentiel d'un univers de jeu roguelike. Bien d'autres idées pourraient être implémentées.

Le tutoriel Roguesharp a été d'une grande aide afin d'obtenir une version jouable assez rapidement, ce qui encourage pour la suite de la programmation.

Le travail en binôme s'est également bien passée et la répartition des tâches fut pertinente.

ROGUEPUNK 2021



Name: Rogue
 Niveau: 1
 Exp : 12/25
 Health: 42/50
 attack: 7 (70%)
 Defense: 6 (40%)
 Credit: 33
 Classe: Cyborg offensif
 Gate : True
 (P)Kit de Reparation : 2

R: Rogue
 R: Soldier Robot

Blindage renforce !! Def +1
 Rogue a ouvert une porte
 Rogue a ouvert une porte
 Rogue a ouvert une porte
 Rogue a ouvert une porte
 Rogue a ouvert une porte
 Teleporteur deverouille !!
 Rogue a ouvert une porte
 Rogue a ouvert une porte

ROGUEPUNK 2021

Soldier Robot et Rogue vont combattre !
 COMBAT
 Vous devez choisir votre action ! A pour attaquer, D pour defendre, F pour fuir et P pour potion !
 Une fois que vous avez choisi vous devez cliquer sur la touche Entree

Vous avez choisi d'attaquer
 Vous avez mis un coup critique !! Degats x2

Nombre de degats infliges par Rogue : 7
 Nombre de degats infliges par l'ennemi : 1
 Nombre de pv restants de l'ennemi : 21

Vous avez choisi de defendre
 Nombre de degats bloques par Rogue : 6
 Nombre de degats infliges par l'ennemi : 1
 Nombre de pv restants de l'ennemi : 21

✦

Vous avez choisi de prendre une potion
 Nombre de degats infliges par l'ennemi : 1
 Nombre de pv restants de l'ennemi : 21

Vous avez choisi d'attaquer
 Nombre de degats infliges par Rogue : 7
 Nombre de degats infliges par l'ennemi : 1
 Nombre de pv restants de l'ennemi : 14

Vous avez choisi de fuir
 Vous n'avez pas pu fuir !!

La defense a echoue
 Nombre de pv restants de l'ennemi : 14

Name: Rogue
 Niveau: 1
 Exp : 12/25
 Health: 48/50
 attack: 7 (70%)
 Defense: 6 (40%)
 Credit: 33
 Classe: Cyborg offensif
 Gate : True
 (P)Kit de Reparation : 1

R: Soldier Robot

Rogue a ouvert une porte
 Rogue a ouvert une porte
 Rogue a ouvert une porte
 Teleporteur deverouille !!
 Rogue a ouvert une porte
 Rogue a ouvert une porte
 Le combat se lance !
 Rogue consumes a Kit de Reparation and recovers 20 health
 Rogue a choisi de fuir

