V1.0 Projet Cosmic Yonder

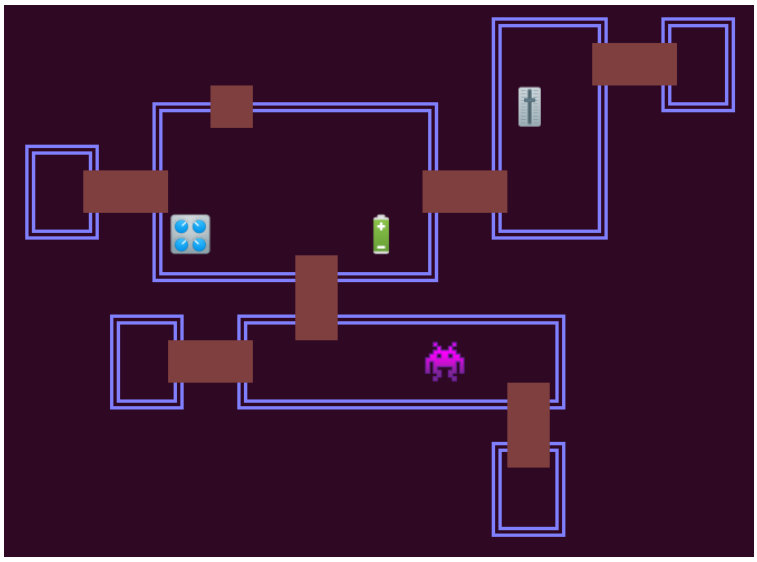
CLASSE préING1 • 2023-2024
AUTEURS Eva ANSERMIN - Romuald GRIGNON
E-MAILS eva.ansermin@cyu.fr - romuald.grignon@cyu.fr

DESCRIPTION DU PROJET	Le but de ce projet est de réaliser un jeu 2D qui intègre une génération procédurale de salles. Vous incarnez un pionnier de l'espace à bord de sa station spatiale parcourant l'immensité du vide. Vous devez réaliser des tâches de maintenance de votre station tout en combattant des menaces biologiques d'origine inconnue. Ce projet met l'accent sur la génération procédurale et sur l'affichage sur le terminal. Ce projet peut s'avérer relativement difficile pour certain(e)s : lisez attentivement les critères du cahier des charges avant de vous engager. Avec une bonne organisation et un bon découpage des parties du code, ce projet est tout à fait faisable si vous êtes motivé(e)s.
INFORMATIONS GENERALES	**Taille de l'équipe** Ce projet est un travail d'équipe. Il est autorisé de se réunir en groupe de 3 au maximum. Si le nombre total d'étudiants n'est pas un multiple de 3 et/ou si des étudiants n'arrivent pas à constituer des groupes, c'est au chargé de TD de statuer sur la formation des groupes. Pensez-donc à anticiper la constitution de vos groupes pour éviter des décisions malheureuses. **Démarrage du projet** Vous obtiendrez de plus amples informations quant aux dates précises de rendu, de soutenance, les critères d'évaluation, le contenu du livrable, …, quand le projet démarrera officiellement. **Dépôt de code** Vous devrez déposer la totalité des fichiers de votre projet sur un dépôt central Git. Il en existe plusieurs disponibles gratuitement sur des sites web comme github.com ou gitlab.com. **Rapport du projet** Un rapport écrit est requis, contenant une brève description de l'équipe et du sujet. Il décrira les différents problèmes rencontrés, les solutions apportées et les résultats. L'idée principale est de montrer comment l'équipe s'est organisée, et quel était le flux de travail appliqué pour atteindre les objectifs du cahier des charges. Le rapport du projet peut être rédigé en français.

	Démonstration Le jour de la présentation de votre projet, votre code sera exécuté sur la machine de votre chargé(e) de TD. La version utilisée sera la dernière fournie sur le dépôt Git avant la date de rendu. Même si vous avez une nouvelle version qui corrige des erreurs ou implémente de nouvelles fonctionnalités le jour de la démonstration, c'est bien la version du rendu qui sera utilisée. **Organisation** Votre projet complet **devrait (dans l'idéal)** être stocké sur un dépôt git (ou un outil similaire) tout au long du projet pour au moins trois raisons : éviter de perdre du travail tout au long du développement de votre application, être capable de travailler en équipe efficacement, et partager vos progrès de développement facilement avec votre chargé de projet. De plus il est **recommandé** de mettre en place un environnement de travail en équipe en utilisant divers outils pour cela (Slack, Trello, Discord, …).
CRITERES GENERAUX	• Le **but principal** du projet est de fournir une **application fonctionnelle** pour l'utilisateur. Le programme doit correspondre à la description en début de document et implémenter toutes les fonctionnalités listées. • Votre code sera généreusement **commenté**. • Tous les éléments de **votre code** (variables, fonctions, commentaires) seront écrits dans la **même langue**. Langue anglaise conseillée mais pas obligatoire. • Votre application ne doit **jamais** s'interrompre de manière intempestive (crash), ou tourner en boucle indéfiniment, quelle que soit la raison. Toutes les erreurs doivent être gérées correctement. Il est préférable de d'avoir une application stable avec moins de fonctionnalités plutôt qu'une application contenant toutes les exigences du cahier des charges mais qui plante trop souvent. Une application qui se stoppe de manière imprévue à cause d'une erreur de segmentation ou d'une exception, par exemple, sera un événement très pénalisant. • Votre application devra être **modulée** afin de ne pas avoir l'ensemble du code dans un seul et même fichier par exemple. Apportez du soin à la conception de votre projet avant de vous lancer dans le code. • Le livrable fourni à votre chargé(e) de TD sera simplement l'**URL** de votre **dépôt Git** accessible **publiquement**. Même si vous n'avez pas utilisé ce dépôt régulièrement au cours du projet, le code final sera livré dessus.
FONCTIONNALITES DU PROJET	• Lorsqu'une partie démarre, un nombre est demandé à l'utilisateur par le programme : c'est la **graine** du niveau. Cette graine servira pour toute gestion des tirages aléatoires au cours de la partie. Elle doit être stockée quelque part en mémoire pour être réutilisée. • Le programme doit déterminer aléatoirement un nombre de salles et allouer l'espace mémoire nécessaire. Le nombre maximum de salles doit pouvoir être modifiable facilement dans votre code. • Par défaut, les salles ne sont pas créées au démarrage du programme, seuls les emplacements mémoire nécessaires sont présents mais aucune information n'est encore présente. C'est au fur et à mesure de la découverte de la carte que les salles vont se créer en mémoire et seront visibles à l'écran. Au départ seule la première salle est réellement créée et visible et votre personnage est placé au centre de cette première salle.

- Une salle est un rectangle de dimensions aléatoires MxN (y compris les murs qui la composent). Une salle fait donc au minimum 3x3 cases pour avoir au moins un espace d'une case au milieu des 4 murs. Les dimensions maximales d'une salle sont à définir librement.
- Chaque salle dispose au minimum d'une porte, et au maximum d'une porte par côté, ce qui fait donc un maximum de 4 portes par salle. Il est donc possible que certains murs ne possèdent pas de porte.
- La première salle dispose toujours d'une porte par côté pour laisser le champ libre dans les 4 directions. C'est la salle de départ.
- Une porte est positionnée à la place d'un bout de mur, sur un côté de la salle, pas obligatoirement au centre de ce côté : elle peut se trouver sur un bord ou un autre du côté, en fonction du tirage aléatoire.

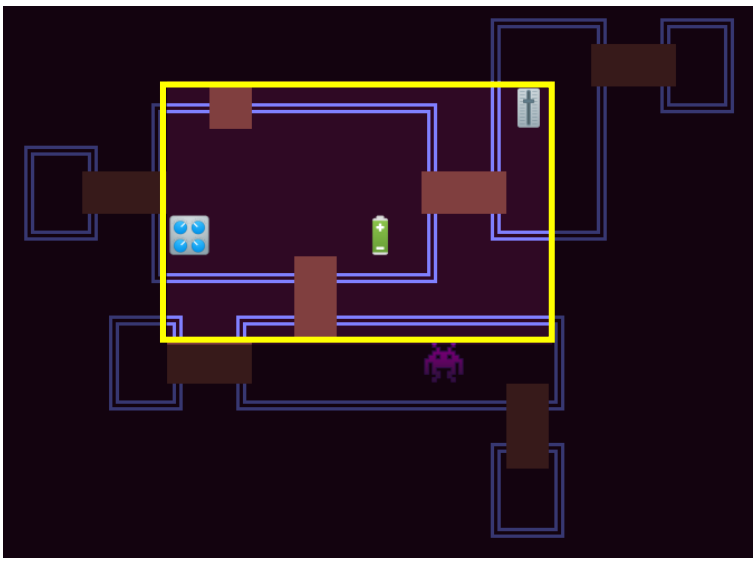
- 2 salles côte à côte ne sont reliées que si une porte se situe entre les 2. Il ne peut pas y avoir de superposition de salles. Votre programme doit donc veiller à ce que cela n'arrive jamais. Pour simplifier l'affichage et la gestion des coordonnées, il est envisageable de dire qu'une porte est en fait située sur 2 cases, une case en périphérie de la première salle, et une autre case en périphérie de la deuxième salle. Ces deux salles étant côte à côte, il y aurait donc 2 murs qui les séparent. Ci-dessous un exemple possible d'affichage de salles :



Ici les portes sont plutôt des couloirs qui relient les salles entre elles

- Une salle est créée quand le personnage principal se déplace sur une porte (voir plus loin les critères concernant les déplacements). A ce moment-là, le programme va effectuer une création de salle aléatoire et l'ajouter dans la liste des salles décrite précédemment. La salle créée va contenir potentiellement des portes : ces portes correspondent forcément à de futures salles non encore générées à ce stade. Votre programme doit donc s'assurer qu'il reste de la place dans le tableau des salles en mémoire. Si ce n'est pas le cas (cela veut dire par exemple ici que nous sommes en train de générer la dernière salle possible) alors la salle générée n'aura aucune porte (aucune autre que celle par laquelle nous venons d'entrer).

- Lorsqu'une salle est créée, s'il reste suffisamment de places pour stocker de futures salles, un nombre aléatoire de portes est généré et les portes sont placées. Au moment où une porte est placée, alors qu'elle n'est pas encore visitée, une information dans le tableau des salles doit être notée pour indiquer que ce créneau mémoire est réservé. Ainsi il ne sera pas possible d'avoir des portes derrière lesquelles nous ne pourrions pas allouer de salle. Ceci permettra d'avoir un labyrinthe de salles toujours cohérent (pas de porte derrière laquelle il n'y aurait « rien »).
- Lorsqu'une salle est générée, si ses dimensions font qu'elle se superpose avec une autre salle, alors il faudra modifier cette dimension pour supprimer ce problème. Aucune superposition n'est possible. Il est possible pour vous d'implémenter une version de code pour laquelle, les salles ne seraient séparées que par 1 seul mur au lieu des 2 cités précédemment : cela pourrait être plus complexe à gérer mais plus agréable visuellement. Vous êtes libres sur ce point. Dans cette hypothèse, 2 salles peuvent être juxtaposées et se partager le même mur de séparation, mais aucune superposition n'est autorisée malgré tout.
- Lors de la génération d'une salle, comme il faut pouvoir faire un programme procédural, il ne faut pas utiliser de l'aléatoire pur : il faudra utiliser la graine indiquée au départ en plus de la position de la porte activée, de l'orientation de la porte activée, et/ou d'autres informations que vous jugerez utiles pour générer une graine servant à la génération de votre salle. Ainsi, à chaque fois que vous lancerez le jeu avec votre graine, et que vous activerez les portes dans le même ordre, vous aurez toujours le même rendu. Cette propriété, en plus d'être un élément important du cahier des charges, vous permettra de débugger plus facilement.
- Peu importe le nombre de salles déjà découvertes ou non, l'écran ne doit afficher qu'une zone réduite autour de la position du joueur. C'est donc une vision relative qui doit se faire à l'écran. Ci-dessous un exemple de ce qui pourrait être affiché à l'écran (zone encadrée en jaune) le reste n'étant pas affiché sur l'écran :



On cherche à n'afficher qu'une partie de la carte en fonction de la position du joueur et les dimensions de la zone d'affichage

- Cette fonctionnalité d'affichage est un autre critère central du projet : il va vous falloir effectuer des changements de repère entre la position de votre joueur dans le jeu, pour que cette position devienne la position centrale de l'affichage : il ne restera plus qu'à convertir les positions absolues des salles en position d'affichage.

- Le personnage peut se déplacer dans les 4 directions orthogonales sur la carte. Il ne peut pas se superposer avec un mur. Il peut se superposer avec une porte par contre. Au moment où le joueur se superpose avec une porte, si la salle « suivante » n'est pas déjà créée, elle le sera comme indiqué précédemment dans ce document. Si elle l'est déjà, aucune action spéciale ne sera faite.
- Le personnage possède à minima les caractéristiques suivantes : points de vie, points d'expérience, et un inventaire (à définir librement).
- Il faudra ajouter sur la carte des éléments avec lesquels interagir (objets à ramasser, entités à combattre, appareils à réparer, …) : chaque action agissant sur les points de vie et/ou les points d'expérience. L'inventaire étant une structure en mémoire vous permettant de gérer les mécaniques du jeu (ramasser un outil utilisable sur un autre objet, …). Ces éléments sont ajoutés de manière procédurale, donc chaque salle doit toujours avoir les mêmes caractéristiques et contenir les mêmes entités si elle est générée à partir de la même graine.
- Les points de vie, d'expérience et le contenu de l'inventaire doivent toujours être affichés à l'écran quelque part autour de la carte.
- Si les points de vie du personnage tombent à zéro, le personnage recommence à la position initiale de départ, mais il conserve ses points d'expérience (en tout ou partie). L'état de la carte à ce moment-là reste inchangé : ce qui a été visité le reste, ce qui est encore inconnu le reste également, l'état des différents objets, monstres, etc… peut être réinitialisé ou non, au choix.
- Le programme doit également gérer une notion de temps : soit un timer qui décompte l'arrivée d'une catastrophe imminente, soit une jauge d'oxygène qui décroit, et c'est la raison pour laquelle le personnage doit effectuer des opérations de maintenance dans la station, en tout cas, cette information doit décroitre en fonction du temps. Cette information doit également être visible à l'écran.
- Votre programme doit intégrer au moins une condition de victoire (fin des tâches à accomplir, etc…) et doit intégrer au moins une condition de défaite (timer échu, oxygène épuisé, …)

- Ce programme démarre par un menu qui permet d'effectuer diverses actions comme démarrer une nouvelle partie, charger une partie sauvegardée précédemment dans des fichiers, ou quitter le programme.
- Lorsque l'on démarre une nouvelle partie, le nom du joueur est demandé. Ce nom sera utilisé pour sauvegarder la partie dans un fichier (pour pouvoir retrouver la sauvegarde). Une partie peut être sauvegardée à n'importe quel moment.
- Si une partie est sauvegardée, le temps de jeu restant doit être cohérent lorsque l'on restaure cette partie.

	• Le programme doit permettre de revenir au menu de départ à n'importe quel moment. • Lorsque l'on souhaite restaurer une sauvegarde, le nom du joueur sera demandé, et la sauvegarde associée sera rechargée en mémoire. • Le programme doit pouvoir fonctionner sans s'arrêter peu importe les fonctionnalités demandées : exemple : on démarre une partie, on la sauvegarde, on la stoppe, on commence une autre partie, on sauvegarde cette autre partie, on stoppe et on recharge la première partie, …
RESSOURCES UTILES	**Github** • www.github.com • https://docs.github.com/en/get-started/quickstart/hello-world **Modification des couleurs (police et fond) dans le terminal** • http://sdz.tdct.org/sdz/des-couleurs-dans-la-console-linux.html **Affichage de caractères dans le terminal** • Bordures : https://www.w3.org/TR/xml-entity-names/025.html • Emojis : https://unicode.org/emoji/charts/full-emoji-list.html