Rapport de soutenance

Soutenance 1

WizzStudio

Clément Bonaud, Mathilde Pujol-Sillié, Mathilde Hubert, Anthon Nazon, Théo Privat

Plan:

Détails de la partie 6:

1. Création de l’environnement python avec pygame pour afficher des formes géométriques
2. Mise en place du projet à l'aide de classes (POO)
3. Création de la partie serveur et client séparé (GameState)
4. Multijoueur avec connection sockets
5. Mise en place du github avec readme
6. Gamemanager pour une logique centralisée
7. Base des pnj et ennemis
8. Caméra
9. EntityManager
10. Sort lançable par le joueur via bouton de la souris (gestion des événements)
11. Murs
12. Hitbox
13. Collisions
14. Système d’ui dynamique
15. Tentative interpolation
16. Événement de collisions
17. IA
18. Système de vie/dégât
19. Collision ennemi/PNJ avec mur
20. IA manager
21. Déplacement via pathfinding
22. Base/prototype pour reconnaissance vocale
23. Animateur/sprite
24. Optimisation du pathfinding(en cours)
25. Pathfinding bugs (en cours de résolution)
26. Carte
27. Logique de tilemap
28. Website
29. Ressources graphiques
30. L’histoire

7. Les annexes:

– Ressources graphiques
– Les captures d’écran

Contexte du projet

- Origine et nature du projet

Au début de notre projet, nous avons hésité sur le type de jeu, notamment entre rôle playing game et incremental game. Incremental game est un type de jeu dans lequel le joueur doit faire progresser une ressource, généralement de l'argent, pour ensuite acheter des améliorations lui permettant de gagner encore plus de cette ressource. Cela va généralement être automatique, il va même y avoir des revenus passifs.
Cependant lors de la mise en accord de la première réunion pour savoir vers quel jeu nous voulions nous tourner, le groupe a décidé de faire un RPG qui pourrait avoir plus de possibilités qu'un incremental game.
Une autre hésitation nous a touché : si nous allions faire un jeu en 2D ou bien en 3D. Si nous partions sur de la 2D nous utiliserions pygame alors que pour la 3D, deux choix s'offraient à nous, Ursina ou Panda3D.
En vérifiant les ressources de ces deux moteurs de jeux, au sein du groupe nous avons trouvé Panda3D un peu plus complexe à utiliser. Nous n'avions donc plus que deux choix possibles. La 3D ou la 2D ; nous sommes partis vers de la 2D, ce qui nous a semblé plus accessible à tous.
Pour le sujet du projet, dès le début nous savions que nous voulions nous orienter vers le monde de la magie mais c'était encore trop vague. C'est effectivement un domaine tellement vaste qui demande quelques spécificités pour se centrer vers un point et ne pas partir dans tous les sens.
Une fois le type de jeu et l'univers décidés, nous avons commencé à noter quelques idées qui nous passaient par la tête pour se mettre d'accord sur une seule et même histoire afin de pouvoir développer ces idées plus tard si besoin.
Suite à cela, il a fallu trouver un nom qui fonctionne avec le jeu mais aussi qui plaise au groupe. Le jeu s'est donc appelé "An bra da cadavra" qui ne plaisait pas forcément au groupe entier. Cependant, un manque d'inspiration à fait que pendant une assez courte période, notre jeu portait ce nom. Mais lors de l'une des réunions, une question est revenue sur le nom initialement donné. Ce qui a mené à de nouvelles propositions. Dont l'une d'elle, qui a fait l'unanimité parmi le groupe, "Avada-Kedavoix" qui est désormais le nom officiel de notre projet.

Objectifs du projet

- Intérêts du projet

  Ce projet nous permettra d'atteindre de multiples objectifs, tous importants pour notre carrière professionnelle. Il va nous aider à nous améliorer professionnellement et techniquement. Concrètement, le but final de ce projet est de créer un jeu pour ordinateur intégrant un système d'intelligence artificielle et un système réseau. Nous réalisons ce projet pour apprendre de nouvelles compétences techniques, par le développement. De plus, par la communication et l'organisation en équipe dans un objectif commun, ce projet va nous apprendre des compétences sociales.

- Objectifs techniques

  Pour ce projet nous avons comme objectifs techniques, la mise en place d'un système d'intelligence artificielle, d'un système de réseau, d'un système de physique avec des collisions, d'un système de reconnaissance vocale et d'autres ajouts mineurs. Dans un premier temps nous nous consacrerons à la mise en place du système de multijoueur via le réseau. Ce système constitue le squelette de l'intégralité du jeu car toute action du joueur et du monde autour de celui-ci devra être adaptée pour ce système. Cette partie constitue le squelette car elle doit permettre à tous les joueurs d'être synchronisés. L'objectif d'un tel module est d'apprendre à organiser une architecture plus complexe qu'un simple jeu sans cette partie multijoueur. Nous rajouterons de la physique au monde, c'est-à-dire que pour empêcher le joueur et autres objets de traverser des murs ce module doit être créé. Il doit aussi détecter les collisions entre des objets et le joueur. Les créatures devront être animées d'une intelligence artificielle pour gérer leur déplacement et leur comportement vis-a-vis du joueur comme les ennemies ou les villageois. Pour prolonger l'expérience utilisateur, l'objectif est de développer un système d'équipement et de progression pour augmenter la difficulté au cours du temps. Et pour finir, pour nous démarquer des autres jeux de ce genre nous devrons développer un système de reconnaissance vocale. Ce système marche grâce à des outils fonctionnant via des intelligences artificielles.

- Objectifs personnels

  Tous ces objectifs ont pour but de nous améliorer dans l'exploitation d'un langage de programmation. Par exemple grâce aux recherches que nous devons faire pour trouver des explications et définitions dans des documentations. Ceci va nous apprendre à savoir où chercher des informations quand on est perdu et améliorer notre compréhension du code.

Nous apprendrons aussi à gérer un projet de son architecture technique à son architecture dans le code. Ceci implique aussi l'apprentissage des conventions de programmations pour assurer une cohérence dans les lignes écrites entre tous les membres du groupe. Au-delà des aspects techniques, ce projet vise également le développement de compétences sociales, notamment la capacité à travailler en équipe, à communiquer efficacement et à s'organiser autour d'un projet commun comportant des tâches variées.

Le projet

Avada-Kedavoix est un RPG où chaque personnage incarne un magicien en 2D. Les joueurs doivent unir leurs pouvoirs et leurs voix pour repousser les ennemis. Vous incarnez un Écho-sorcier et apprenez à canaliser la magie à travers votre voix, plus votre maîtrise est grande, plus vos sorts deviennent puissants. Affrontez les ténèbres dans la Tour des Abysses, une structure inversée plongeant directement dans les Enfers. Coopérez avec d'autres joueurs pour affronter des créatures infernales : démons, zombies, squelettes et autres abominations qui hantent les profondeurs de la tour. Avancez dans le labyrinthe, prenez garde à ne pas vous égarer. Vous devrez pour progresser, discuter et marchander des armes et des denrées dans le village avec les différents marchands et personnages non-joueurs. Il n'y a aucun affrontement entre joueurs, seule la magie collective permet de triompher du mal.

Notre jeu est un RPG, signifiant role play game, dans ce type de jeu le joueur incarne un personnage fictif et très souvent son but est d'avancer dans son aventure en éliminant des ennemis de plus en plus fort le tout en s'améliorant pour progresser toujours plus loin.

Dans notre jeu, le but est d'affronter des ennemis pour progresser. Mais sa particularité c'est la présence de la Tour des Abysses, il s'agit d'un donjon composé de plusieurs étages. Dans chaque étage se trouvent des monstres mais aussi des pièges. Pour progresser dans cette tour, le joueur va devoir utiliser sa voix pour lancer des sorts.

Qui sommes-nous ?

- Notre entreprise

  Nous sommes Wizz Studio, une entreprise de création de jeux vidéo. Nous sommes actuellement en train de développer un nouveau jeu vidéo nommé Avada-Kedavoix.
  Wizz à été choisi pour le mot wizard (en anglais sorcier) qui est un thème récurrent pour tous nos jeux actuels et futurs, en cours d'élaboration ou simplement à l'état de simples idées pour le moment. Ce nom est en anglais afin de pouvoir toucher un public bien plus large que celui de la France. Dans le jeu Avada-Kedavoix, le ou les joueurs incarnent ces sorciers.

- Notre équipe

  Nous sommes une équipe de 5 personnes dont:

  - Clément Bonnaud, le chef de projet : il est donc responsable de l'organisation et de surveiller l'avancement du projet. Il guide et rassemble les membres, organise les réunions et s'assure que nous rendons bien tout le travail lorsqu'il faut. Il travaille beaucoup sur la partie intelligence artificielle et les recherches algorithmique notamment A*. Il s'occupe aussi de la partie physique et traitement vocal grâce à son expérience en développement de jeu python.

  - Mathilde Pujol-Sillié, la responsable des rendus écrits et des comptes rendus de réunions. Elle s'assure de garder une trace claire et normalement sans fautes d'orthographe de tout ce qui a été fait ou prévu. Elle est également responsable de tout le côté rédaction donc majoritairement des comptes rendus des cahier des charges, ainsi que de la relecture de tout ce qui est diaporama par exemple, pour essayer au maximum d'évincer les erreurs d'orthographe. Ce qui correspond bien à son profil, puisqu'elle aime écrire. Elle est en charge de toutes les parties dialogues et histoires du jeu et de toutes les interfaces liées à l'histoire. Et aussi d'une partie du système d'intelligence artificielle, et du site web et de la coloration de certains graphismes.

  - Mathilde Hubert, la responsable du temps ainsi que la graphiste du projet. Elle veille à ce que l'avancement du projet respecte les délais impartis, c'est grâce à elle que nous arrivons à avoir une idée de où

est-ce que nous en sommes, des objectifs qu'il faudrait atteindre, sur quels points nous sommes en avance ou en retard. Elle s'occupe également de concevoir certains graphismes, tels que les sorts, en s'appuyant sur ses compétences en dessin. Elle a déjà utilisé le langage Python lors de la réalisation de projets en équipe au lycée.

- Anthon Nazon, la personne qui s'occupe de la partie communication du jeu. Il s'occupe de la présentation du jeu, et s'assure de la communication des informations à l'intérieur du groupe et de parler de nos projets et jeux, il explique notre jeu pour donner envie aux gens de jouer. Il s'occupe également de la physique du jeu et des cartes. Il participe également à la conception des cartes du jeu en s'associant avec la graphiste du groupe. Et il cherche également certaines bibliothèques de modèles graphiques déjà existant afin de nous faciliter quelques tâches.

- Théo Privat, le développeur en chef du projet. C'est le membre du groupe avec la plus grande expérience, il a configuré le GitHub du projet et toutes les explications pour aider les autres membres. Il conçoit l'architecture centrale du jeu afin de fournir une base solide pour l'ensemble des fonctionnalités. Il maîtrise le langage Python et les concepts clés du développement de jeux vidéo, tels que le réseau, la physique et l'intelligence artificielle. Il possède également une expérience dans la création de jeux vidéo acquise lors du développement d'un jeu mobile avec Unity.

- Notre façon de travailler

Nous travaillons préférentiellement de manière individuelle, en se répartissant les tâches, en fonction de nos compétences mais aussi des aspects et techniques qui nous intéressent le plus d'apprendre. Certaines tâches sont réalisées en binôme voire certaines fois en trinôme par soucis de temps principalement mais aussi pour varier les idées et les savoirs afin d'obtenir le jeu qui correspond au maximum à la vision de chaque personne. Beaucoup de méthodes dont nous avons besoins n'étant pas enseignées en cours, nous nous appuyons sur des ressources extérieures, principalement des leçons et des vidéos en ligne mais aussi en demandant à d'autres personnes plus expérimentées, pour trouver ce dont nous avons besoin. Afin que personne ne soit livré à son sort, nous nous assurons d'organiser régulièrement des réunions pour que tout le monde sache ou est-ce que

nous en sommes et quels sont les points qui posent un problème. Nous utilisons github afin de mettre tout notre travail en commun.

Organisation

- Méthodes d'organisation

  Afin de bien nous organiser pour la réalisation de ce projet de groupe, nous avons choisi d'utiliser l'application Miro, qui nous permet de suivre l'avancement des tâches en temps réel, de noter nos prévisions pour les prochaines soutenances, de visualiser la carte mentale de notre jeu, ainsi que de définir nos objectifs pour cette première soutenance.

  De plus, pour chaque document devant être rédigé, tels que le cahier des charges fonctionnelles ou ce rapport de soutenance, nous avons créé des documents partagés. Cela permet à chacun de rédiger sa partie tout en tenant compte du travail des autres, mais aussi de s'entraider si nécessaire sur certaines parties.

  Pour faciliter la communication au sein de l'équipe, nous avons décidé de créer un groupe sur Instagram, ainsi qu'un autre sur Discord. Cela nous permet d'échanger rapidement et de nous entraider en cas de problèmes.

  Enfin, nous avons organisé des réunions régulièrement afin de suivre l'avancement de la conception du projet et de discuter ensemble des points importants. À la suite de chaque réunion, nous avons rédigé un compte rendu afin de garder une trace des points abordés ainsi que des modifications décidées, que ce soit en termes d'organisation ou concernant le fonctionnement du jeu.

- Planning

  Au début de l'année, nous avons créé un planning sur lequel nous avons noté toutes les dates importantes pour ce projet, ainsi que les dates de nos réunions. Vous pourrez trouver ce planning dans les annexes.

- Étapes importantes

  Au début du projet, il était essentiel de bien organiser le travail au sein du groupe, notamment en ce qui concerne la répartition des tâches. Nous avons donc organisé une réunion durant laquelle nous avons fait une liste de toutes les tâches à réaliser pour le projet.

Nous avons ensuite essayé d'évaluer la difficulté et la charge de travail que représentait chacune de ces tâches. La répartition s'est faite collectivement, en tenant compte des compétences de chacun, mais aussi de l'envie d'apprendre et de découvrir de nouveaux aspects du projet. Par exemple, Mathilde étant particulièrement à l'aise avec l'écriture, nous avons décidé de lui confier la rédaction du storyboard.

Chaque membre du groupe est ainsi devenu responsable d'une tâche principale, tout en étant suppléant sur au moins une autre. Cela nous permet de nous entraider en cas de difficulté ou d'absence.

Enfin, après la répartition des tâches, nous avons mis en place un suivi régulier grâce à l'application Miro, afin de vérifier l'avancement du projet et d'ajuster l'organisation si nécessaire. Nous pouvons ainsi voir si une tâche est terminée, en cours de réalisation ou non commencée et finalement, nous pouvons nous assurer que le projet avance dans les délais impartis.

Présentation technique dans l’ordre chronologique:

1. Environnement python avec pygame: Pour avoir une idée de ce que nous allions faire, nous avons commencé par créer une scène avec pygame vierge. Cette scène a été remplie de formes géométriques de différentes couleurs pour commencer à avoir une idée de comment marcher pygame. Ceci nous a permis de nous projeter pour savoir quelle architecture prendre pour le développement. Nous avons créé un personnage simple, représenté par une sphère qui se déplace grâce aux touches du clavier. Cet environnement était simple et minimaliste mais nous l’avons fait évoluer.

2. Nous avons choisi d’utiliser l’approcher en POO ou programmation orientée objet. Ce paradigme est très utile dans la création de jeu car elle permet d’isoler des fonctionnalités dans des classes ou chacune a un rôle bien précis. Ceci permet de créer plusieurs objets/entités avec la même logique simplement. Par exemple, tous les joueurs possèdent la même classe joueur simplifiant l’architecture du code. Nous avons créé des “Manager” qui s'occupent de gérer une partie précise de la logique, par exemple le networkManager qui s'occupe du réseau. Ce paradigme bien qu'utile à poser un problème, tous les membres de l’équipe ne connaissaient pas cette manière de coder et ils ont dû donc apprendre pour pouvoir comprendre ce qui avait été fait et par la suite comment participer au projet.

3. Nous avons par la suite commencé le multijoueur. En s'aidant de contenu sur YouTube nous avons créé un serveur sockets qui s'occupe de gérer les connexions. Nous avons aussi rajouté la partie pour que les clients rejoignent le serveur. Mais concrètement rien ne s'est passé, il a donc fallu que le serveur et le client communique. Pour se mettre d'accord, le serveur et le client se partagent une copie du monde, le “GameState”. Dans cette copie se trouve l’ensemble des éléments du jeu comme tous les joueurs, tous les pnj et tous les ennemis sont listés. Pour la protection de ces données seul le serveur a le droit de les modifier. C’est donc 30 fois par seconde que le serveur envoie à tous ses clients l’état du monde. Tous les clients vont par la suite dessiner tous ces éléments à leur place permettant à tous les joueurs de voir un seul et même monde. Cette partie a posé beaucoup de problèmes notamment sur le déplacement des joueurs. Comme seul le serveur a le droit de modifier l’état du monde, les joueurs ne peuvent se déplacer que localement. Pour permettre au joueur de se déplacer aussi du côté du serveur il a fallu que le joueur envoie sa nouvelle position au serveur pour qu’il l’enregistre. Ceci permet à tous les autres joueurs de voir où nous sommes.

4. Pour gérer les requêtes des ajustements ont dû être fait. Par exemple pour simplifier l'envoie et la réception de requêtes, elles sont encapsulées grâce à la classe Message qui lui attribue un type, son payload (l'information qui doit être envoyer) et en header la longueur de la requête. Le header permet d'éviter des erreurs en délimitant l'information et éviter de dépasser sur celle d'après ou même éviter d'oublier un morceau de l'information. Cette longueur est inscrite dans les premiers octets du message. De plus, ces requêtes possèdent un type ce qui va permettre de traiter plus simplement l'information. Par exemple il existe les types: CONNECT, DISCONNECT, PLAYER_UPDATE, PLAYER_CAST_SPELL. Et pour finir il y a le payload ou la partie utilisable. Il s'agit de l'information qui est envoyée, mais pour pouvoir envoyer de l'information via socket il est compliqué et peu recommandé d'envoyer les objets python directement via le réseau. Pour l'envoie du gamestate par exemple, on ne va pas envoyer tous les objets python en même temps. C'est pour cela que nous sérialisons les instances des classes. Pour ce faire chaque classe qui doit transiter entre le client et le serveur se voit hériter de la classe Serializable qui permet d'automatiquement sérialiser les attributs et supprime les méthodes pour ne garder que les valeurs importantes. Le serveur avant d'envoyer le GameState va donc sérialiser, transformer en texte, toutes les instances de classe via la méthode to_dict hériter de la classe Sérializable. Puis le client va faire l'inverse de son côté il va récupérer la requête qu'il va désérialiser via la méthode from_dict pour récupérer l'objet en python à partir du texte et pouvoir le traiter plus simplement. Cette méthode de transformer toutes les instances de classe en json permet de simplifier grandement le développement car elle est universelle. Par exemple, si nous voulons créer une nouvelle classe comme Enemy2 nous n'avons pas besoin de tout refaire mais simplement de le faire hériter de la classe Serializable.

5. Une fois la base de pygame et du multijoueur faites nous avons créer le dépôt github et des fichier README.md, CONTRIBUTING.md et CODE_IMPROVEMENTS.md. Ces fichiers générer grâce à l'ia permet d'aider ceux qui ont du mal à mettre en place un projet en les guidant étape par étape pour l'installer, installer les dépendances et lancer le projet. Ces fichiers permettent aussi d'expliquer comment concrètement participer au projet, en enseignant comment contribuer comment créer une branche pour une fonctionnalité et comment respecter certaines convention comme celle de nommage des variable et le fait de bien documenter le code que nous écrivons. Toutes ces aides permettent de maintenir le projet le plus propre possible ainsi que permettre à tout le monde de comprendre le code des autres sans devoir perdre du temps à le comprendre. Et enfin ces fiches permettent de suivre l'état du projet en listant les fonctionnalités déjà présentes et celles qu'il manque à développer.

6. Pour centraliser la logique du côté du client nous avons créé le GameManager qui centralise tous les éléments du côté client. Il s'agit de la classe la plus complète. Il s'occupe de mettre en commun les requêtes du serveur, de dessiner les éléments, de redistribuer les évènements pygame aux modules qui en ont besoin et de mettre à jour le monde local.

7. Nous avons par la suite créer une base pour les ennemis et pour les pnj vierge pour tester si l'ajout de nouvelles entités complexifie le code. Mais après des ajustements, l'ajout de nouvelles classes se fait assez simplement.

8. Le joueur pouvait se déplacer et les autres clients le voient se déplacer mais la caméra est fixe et le joueur pouvait sortir du champ de vision de la caméra. C'est pour cela que nous avons dû mettre en place la caméra. Mais les caméras en pygame n'existent pas réellement. Nous avons dû rajouter localement un décalage de tous les éléments affichés pour que le joueur soit au centre de l'écran. C'est donc dans le GameManager que chaque fraction de seconde, le manageur va récupérer le décalage du joueur local puis va localement seulement dessiner tous les éléments en rajouter à leur position ce décalage.

9. Pour centraliser les entités dans le GameState nous avons créé l'EntityManager ce module permet de gérer toutes les entités du même type par exemple dans le GameState il y a la liste de tous les joueurs. Cela permet de mettre à jour simplement les entités locales avec celles reçues du serveur. Cela permet aussi de supprimer une entité locale si elle n'est plus présente dans la copie du serveur. De plus, la centralisation des entités permet de toutes les visualiser en un seul bloc.

10. Nous avons ensuite gérer les sorts. Grâce aux événement de pygame nous pouvons lancer un sort avec le bouton de la souris. Les sorts sont gérés comme des entités car elles peuvent mourir quand elle touche un ennemi ou au bout d'un certain temps. Pour ce faire, quand le joueur appuie sur le bouton de sa souris cela va envoyer au serveur une demande de création de sort. Après l'avoir créé, le spell va avancer tout droit dans la direction du pointeur de la souris. La mise à jour de sa position se fait du côté du serveur comme le reste des mise a jour. Les sorts ne sont représentés que par des sphères bleues pour lors de la première soutenance. La création des sorts a permis la mise en place de la gestion des évènements qui sera utile pour la suite.

11. Par la suite nous avons rajouté des murs ceci peut paraître simple mais ceci a été bien plus compliqué que prévu. Nous avons créé une simple forme géométrique mais nous avons dû lui rajouter une hitbox et gérer les collision avec celle-ci.

12. Pour gérer les collisions nous avons créé la class hitbox et nous l'avons ajouté aux entités. Les hitbox se servent de pygame notamment sa méthode spritecollide qui permet de savoir si 2 surfaces de pygame donc 2 éléments se touchent. Cette information sera alors utilisée pour simuler des collisions.

13. Ensuite pour créer les collisions entre le joueur et les murs nous allons à chaque fois que le joueur se déplace regarder s'il se trouve dans la hitbox d'un obstacle et si c'est le cas on le repositionne à sa position d'origine. Ceci a donc permis de délimiter le monde pour tester les collisions et empêcher le joueur de sortir de la carte.

14. Pour simplifier le processus de création de menus et l'interface, tout une méthode de création de menus a été créée. Nous avons centralisé les différents menus dans un seul fichier qui les contiennent tous. Cependant pour créer un menu ou une interface il suffit de suivre un motif dont le contenu du menu doit être une fonction qui renvoie une liste de composants. Chaque composant a été créé manuellement et représente un texte, un bouton, une entrée de texte. Il suffit alors de lister les différents éléments d'une interface pour qu'il apparaisse automatiquement sans devoir manuellement gérer les surface ou autre propriété de pygame. Pour ces composants il y a aussi la possibilité de choisir une position d'ancrage cela correspond a quel coin de l'écran le composant doit être ancré. Concrètement cela permet de placer un texte en haut au milieu de l'écran et la position donnée a cet élément sera relative à ce point. Ceci permet au menu d'être adapté à toutes résolutions. Cet environnement de travail permet aussi de faire des menus dynamiques en permettant de changer un composant pendant que le jeu tourne. En mettant à jour les éléments directement dans la fonction.
    Exemple de composant:

```
# adress_input
TextInput(
    "Ip Address",
    position=(0, -100),
    width=200,
    height=50,
    onTextChanged=lambda x: set_val(0, x),
    anchor=Anchor.CENTER,
    initial_text="127.0.0.1",
),
```

15. Pour fluidifier le mouvement des entités nous avons essayé d'appliquer une interpolation aux entités du côté du client. L'interpolation permet de passer de simplement téléporter l'entité à sa nouvelle position à créer un mouvement local seulement fluide pour masquer la téléportation. Cependant cette partie a été commencée puis mise en suspens car trop complexe et moins importante.

16. Nous avons aussi développé un manager de collision nous permettant d'effectuer des actions spéciales lors de certaines collisions (ex: la collision entre une attaque et un ennemi infliges des dégâts à l'ennemi).
    Ce manager reposent principalement sur deux parties, une première étant une liste de dictionnaire réunissant la totalité des collisions possibles avec avec les deux entités de la collision et la fonction réalisant les opérations de la collision elle permet de faire un lien rapide entre une collision entre deux entités et une action (collisions.py) et la deuxième permet de tester si il y a des collisions et dans ce cas de faire appel à la première partie pour réaliser les collisions (collisionManager.py).

17. Des premières ébauches d'ia ont été réalisées. un pour les PNJs et une pour les ennemis, celle des ennemis se déplacent tout droit vers le joueur le plus proche et celle des PNJs vers une position aléatoire. Cette version était peu lisible et très difficile à comprendre vu qu'elle était intégrée au code des objets des entités.

18. Pour les ennemis et les joueurs nous avons créé un système de vie/dégât, les dégâts du joueurs se transmettent à ses attaques qui sont infligées lors du contact avec un ennemi. Les ennemis ne possédant pas encore d'attaques leurs dégâts sont pour l'instant inutiles. Cette partie a nécessité l'ajout d'une collision (dans collisions.py).

19. Les ennemis et les Pnjs ayant finalement un moyen de déplacement il a fallu faire en sorte qu'ils arrêtent de traverser les murs, nous leurs avons donc ajouté un système de collision avec des murs suivant le même principe énoncé précédemment pour le joueur.

20. Le système d'ia étant complètement brouillon nous nous sommes penchés sur la création d'un manager d'IA (iaManager.py). Ainsi, nous avons conçu des outils simples , comme la récupération des positions des joueurs ,celle du plus proche uniquement ou bien encore le calcul d'une direction grâce à une position.

Avec l'aide de ces outils nous avons pu donner aux Pnj une IA les faisant aller en ligne droite vers une position aléatoire et aux ennemis une IA les faisant se diriger vers le joueur le plus proche. Pour ce faire nous avons créé une nouvelle classe ia permettant de faire le lien entre une entité et son ia (portant le nom de l'entité).

21. Nous avons donc des déplacements qui étaient effectués mais ceux-ci ne prenait pas en compte les murs (les entités rentraient dans les murs et glissaient dessus), ce qui nous a mené à développer et implémenter un système de path finding.
    Le système que l'on a choisis d'utiliser est l'algorithme A*. il reposes sur une évaluation des distance à parcourire pour chacunes des cases visitées (distance connues depuis la position de départ + distance supposée avec un calcul de la distance à vol d'oiseau ou distance de Manhattan) on choisit ensuite de visiter la case avec un distance la plus faible et en cas d'égalité celle la plus proche du point d'arrivée et on ajoutes toutes ses voisines a une liste de case à visiter puis on pioche la suivante dans cette liste en respectant les mêmes critères en en suivant les mêmes action jusqu'à trouver l'arrivée (ou ne plus avoir de cases à visiter).

    (voici les vidéos / documentations nous ayant permis de développer et intégrer cet algorithme:

    - https://fr.wikipedia.org/wiki/Algorithme_A,
    - https://www.youtube.com/watch?v=iMRJzA8BBIg

)

22. Pour la première soutenance il y a aussi un prototype de reconnaissance vocale grâce à vosk. Vosk est une bibliothèque python et qui grâce à un modèle en local permet de transcrire ce que le joueur dit. Cette transcription permet par la suite d'exécuter des actions. Pour la première soutenance il n'y a que le mot "spell" qui permet de lancer un sort. Cette partie constitue l'identité d'Avada-Kedavoix. Celle-ci fonctionne de la manière suivante: quand le joueur parle, le flux de l'audio est ajouté à une file. Cette fille va ensuite être retirée pour extraire les mots retranscrits. Ces mots déchiffrer vont être ajoutés à une nouvelle file globale. Cette file qui contient les paroles du joueur va être triée à chaque mise à jour du GameManager. Le GameManager va alors regarder si les paroles contiennent une des actions enregistrées. Par exemple, si dans les paroles se trouve le mot "spell" le GameManager va déclencher l'apparition d'un sort. Vosk a besoin d'un modèle en local, il en existe de plusieurs tailles et de plusieurs langues mais nous avons pour l'instant utiliser seulement le modèle small en français.

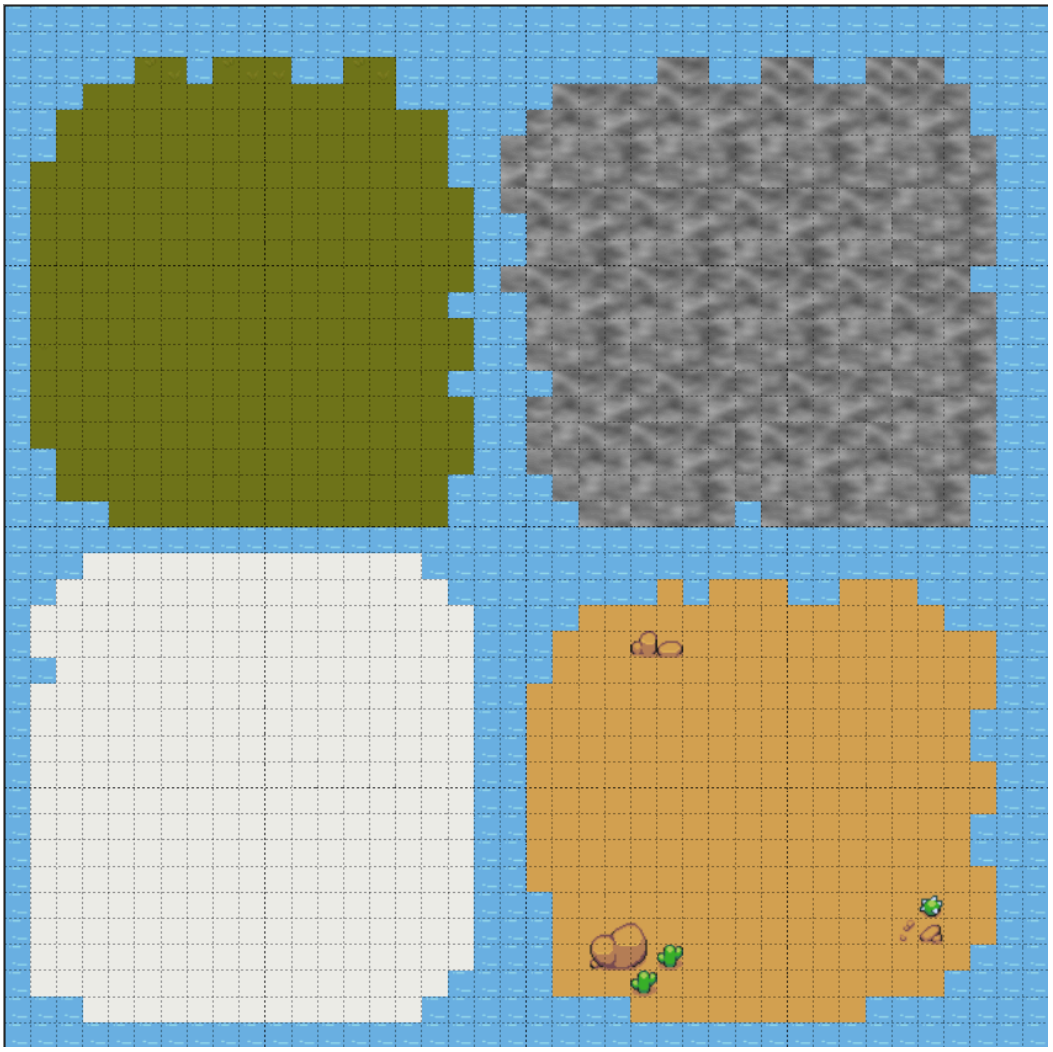
23. Pour ajouter nos graphismes au jeu nous avons d'abord commencé par mettre en place un module permettant de s'occuper des animation du personnage du joueurs. En définissant plusieurs images par animation et en les changeant à intervales réguliers cela donne l'illusion que le personnage marche ou reste sur place.

24. Nous avions un pathfinding fonctionnel toutefois le problème de cette solution est que la recherche d'un chemin nécessite une très grande quantité de calcul il est donc impossible de rechercher tout le chemin d'un coup dans le jeu cela créerait des lags trop importants. et aussi si la cible se déplace pendant qu'il suit le chemin qui vient d'être créé, il n'arrivera pas au bon endroit.
    Nous avons donc pris plusieurs mesures afin d'optimiser l'algorithme. Premièrement nous avons séparé le calculs sur plusieurs frame (ex on calcule 5 position vers la destination puis on se déplace et on recalculé 5 positions) ce qui résout aussi le problème des déplacements de la cible. La deuxième solution est d'espacer les positions nous effectuons avons d'implémenter cette solution une position = 1 pixel, nous avons donc créé une carte séparant chaque position de 20 pixels (modifiable) c'est un système facile à implémenter vu qu'il suffit d'arrondir chacune des coordonnées à 20 près.

25. Malgré tout il nous restes deux bugs:
    - Premièrement: lors du calculs des positions et de l'accessibilité de la position il semblerait qu'il ne trouves de collision avec des murs pour certaines positions (le problème est en cours de résolution nous avons donc désactivés la physique des ennemis afin de régler ce problème).
    - Deuxièmement: comme nous effectuons que 5 calculs lors de déplacement les ennemis risquent de rester bloqué derrière certains murs: ils estiment que le chemin le plus rapide pour passer le mur est en allant vers la droite car la cible se trouve sur leurs droite puis passent à gauche de la cible et donc ils changent d'avis ce qui provoques une oscillation entre deux positions.

26. Pour la map nous avons décidé d'utiliser le tilemapping. Pour ce faire nous nous sommes orientés vers Tiled. Ce logiciel nous permettra d'utiliser des calques afin de superposer les objets et la carte en elle-même et donc créer un effet de profondeur.
    Une fois le logiciel choisi, nous avons commencé à faire des recherches sur des sites comme Craft Pix ou encore Itch.io afin de trouver des ressources graphiques que nous pourrons utiliser.
    Nous avons commencé avec une map uniquement constituée de tuiles représentant de l'herbe pour pouvoir faire des tests.

Finalement, la map est toujours en phase de conception mais elle est plus développée. La taille reste encore à être définie. En effet pour le moment, la map a l'air assez petite. De plus, la taille des tuiles doit être pensée car nous avons des ressources en 32X32 pixels ce qui a été donné comme référence au début mais avec les nouvelles recherches établies, nous possédons des assets en 16X16 pixels. Nous avons donc modifié la map pour qu'elle soit avec des tuiles de 16X16 pixels sauf que les assets de 32X32 pixels ne peuvent plus être utilisés car une tuile de 32X32 pixels se traduit normalement en un bloc de 4 tuiles alors que dans certains cas, ce bloc et représenté uniquement par 2 tuiles collés horizontalement. Ce qui nous empêche de pouvoir être plus précis dans ce que nous voulons faire.



27. Pour gérer l'affichage de la map, nous nous sommes servis de la bibliothèque pytmx qui permet de simplement afficher une carte en .tmx, le format des cartes de Tiled. Il a donc suffit de redessiner chaque tuile pour former une grande carte ou le joueur peut se déplacer. Pour encapsuler cette logique nous avons encore une fois créé une classe spécifique.

28. Pour le site web, nous avons commencé par regarder la documentation sur le site W3Schools afin de revoir le fonctionnement des balises HTML ainsi que les bases du CSS.

    Puis, nous avons conçu la structure de base du site en créant les différentes pages HTML et en les interconnectant entre elles. Nous avons ensuite implémenté un menu de navigation commun à l'ensemble des pages en utilisant les listes HTML.
    En parallèle, nous avons utilisé le CSS pour réaliser un visuel simple, que nous souhaiterions améliorer par la suite.

    Nous avons commencé à intégrer des informations sur chacune des pages même si celles-ci restent encore à compléter. Nous avons notamment inclus un bouton de téléchargement permettant d'accéder à notre dépôt GitHub sur lequel le jeu est hébergé, ainsi qu'au rapport de la première soutenance au format PDF. Nous avons également commencé la rédaction de la biographie de chaque membre de l'équipe.
    Par la suite, nous voudrions ajouter davantage de contenu en rapport avec le jeu tels que la description des ennemis, l'environnement du jeu, l'histoire principale ainsi que d'autres éléments. Nous souhaiterions également rajouter une vidéo de démonstration du jeu à côté du résumé.

    Pour réaliser certains visuels du site web, tels que les cartes de présentation des membres de l'équipe avec un effet de survol, nous nous sommes inspirés de différentes vidéos.
    Voici quelques liens utilisés :
    - CSS Card Hover Effects | HTML & CSS
    - How To Make Animated Website Design Using HTML And CSS Step By Step

    Enfin, pour améliorer le visuel du site, nous avons commencé à réaliser un prototype en utilisant le site Figma. Cela nous permet de visualiser précisément le rendu que nous souhaitons créer avant de passer à l'implémentation du code.

29. Pour les graphismes, nous avions décidé de dessiner nous-mêmes les sorts et les objets de l'inventaire. Concernant les personnages, si le temps nous le permettait, nous souhaiterions également les réaliser nous-mêmes et dans le cas contraire nous utiliserons des ressources trouvées sur Internet.

    Nous avons déjà recherché certaines ressources au cas où cette situation se produirait comme celles que vous pourrez trouver en annexe.

    En revanche, nous avons encore très peu avancé sur les graphismes à réaliser nous-mêmes car Mathilde H. n'a pu ramener sa tablette graphique

que récemment à Toulouse. Cependant, nous avons réalisé un premier prototype de cristal de glace, qui pourra être utilisé dans la création d'un sort de glace.

30. Pour l'histoire, nous avons réalisé tout d'abord des schémas des dialogues, pour être fixés sur quel choix amène à quelle réponse, tout en laissant des options secondaires au cas où ces fonctionnalités ne verraient pas le jour, car nous ne sommes pas encore fixés sur ce point. L'écriture de l'histoire principale avec les explications de fonctionnement du début est presque terminée. Seule la version en français a été réalisée pour le moment, il est prévu qu'il y ait une version en anglais. L'implémentation n'a pas encore été faite, il manque également le cadre dans lequel seront affichés toutes les parties de ce texte.

7. Les Annexes

- Architecture du projet

```
avada-kedavoix/
├── client/                          # Code côté client (client-serveur)
│   ├── main.py                      # Point d'entrée du client
│   ├── clientManager.py             # Gestion de la connexion et des messages réseau
│   ├── gameManager.py               # Gestion globale du jeu (singleton)
│   ├── menus.py                     # Menus (main, join, host)
│   ├── utils.py                     # Fonctions utilitaires
│   ├── classes/
│   │   ├── player.py                # Classe Player avec animations et mouvements
│   │   ├── enemy.py                 # Classe Enemy avec IA
│   │   ├── pnj.py                   # PNJs (personnages non joueurs)
│   │   ├── spell.py                 # Système de sorts avec hitbox
│   │   ├── wall.py                  # Obstacles et murs
│   │   ├── mapBackground.py         # Fond de carte
│   │   ├── animator.py              # Système d'animations
│   │   └── hitbox.py                # Gestion des collisions
│   ├── ui/
│   │   ├── UI.py                    # Gestionnaire d'interface
│   │   ├── button.py                # Composant bouton
│   │   ├── text.py                  # Composant texte
│   │   ├── textInput.py             # Champ de saisie
│   │   └── uiUtils.py               # Utilitaires UI
│   ├── enums/
│   │   └── anchor.py                # Énumération anchor (positionnement UI)
│   ├── voice/
│   │   ├── realtimeVoice.py         # Reconnaissance vocale en temps réel
│   │   └── vosk-model-small-fr-0.22/  # Modèle de reconnaissance vocale français
│   ├── ressources/
│   │   ├── wizzard-test/           # Sprites de magiciens
│   │   └── tiles/                   # Cartes Tiled et tilesets
│   └── __pycache__/
│
├── server/                          # Code côté serveur
│   ├── main.py                      # Point d'entrée, gestion des clients et broadcast
│   ├── NetworkManager.py            # Gestion des connexions socket (singleton)
│   ├── gameState.py                 # État global du jeu
│   ├── message.py                   # Protocole et types de messages
│   ├── collisions.py                # Détection des collisions côté serveur
│   ├── managers/
│   │   ├── entityManager.py         # Gestion des entités (joueurs, ennemis, etc.)
│   │   ├── collisionManager.py      # Gestion des collisions
│   │   └── iaManager.py             # Gestion de l'IA des ennemis
│   ├── ia/
│   │   └── pathFinding.py           # Algorithme de pathfinding
│   ├── classes/
│   │   └── serializable.py          # Classe de base sérialisable
│   └── __pycache__/
│
├── .github/                         # Configuration GitHub
├── README.md                        # Documentation du projet
├── CONTRIBUTING.md                  # Guide de contribution
├── CODE_IMPROVEMENTS.md             # Suggestions d'améliorations
├── requirements.txt                 # Dépendances Python
└── .gitignore                       # Fichiers à ignorer
```

- Ressources graphiques

Ressource graphique trouvée sur Internet représentant un magicien de classe glace.

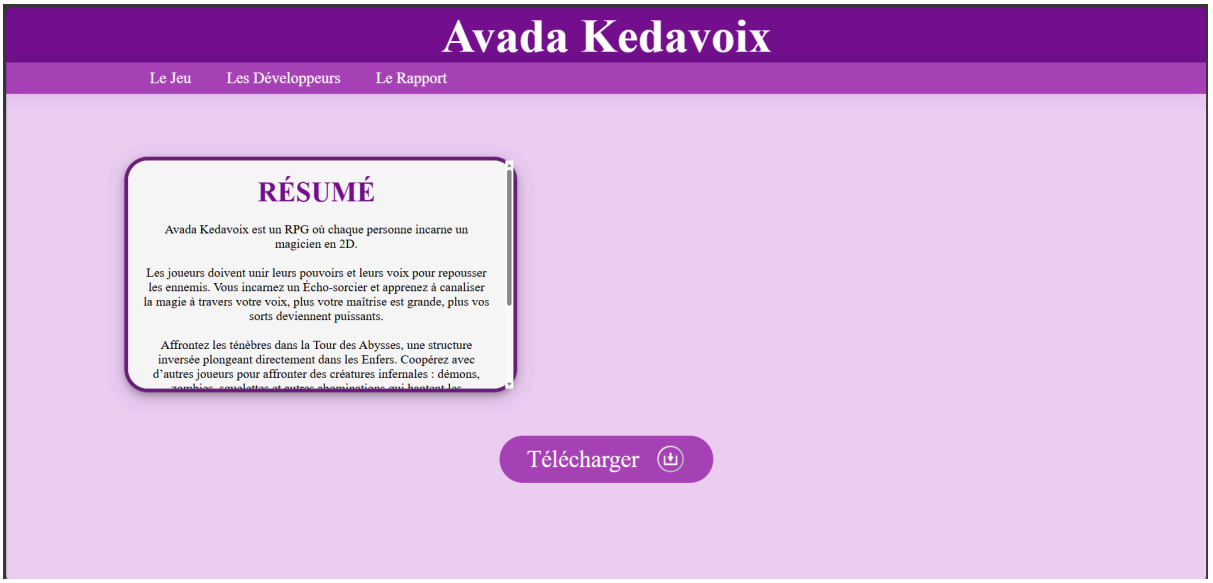
Ressource graphique trouvée sur Internet représentant un magicien de classe feu

Prototype cristal de glace

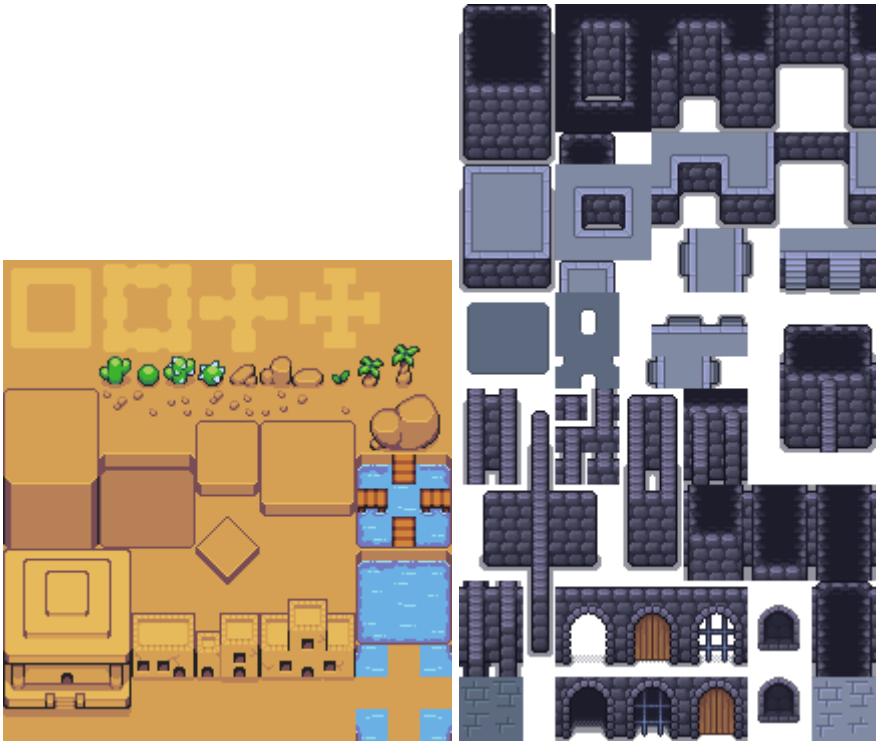
- Le planning

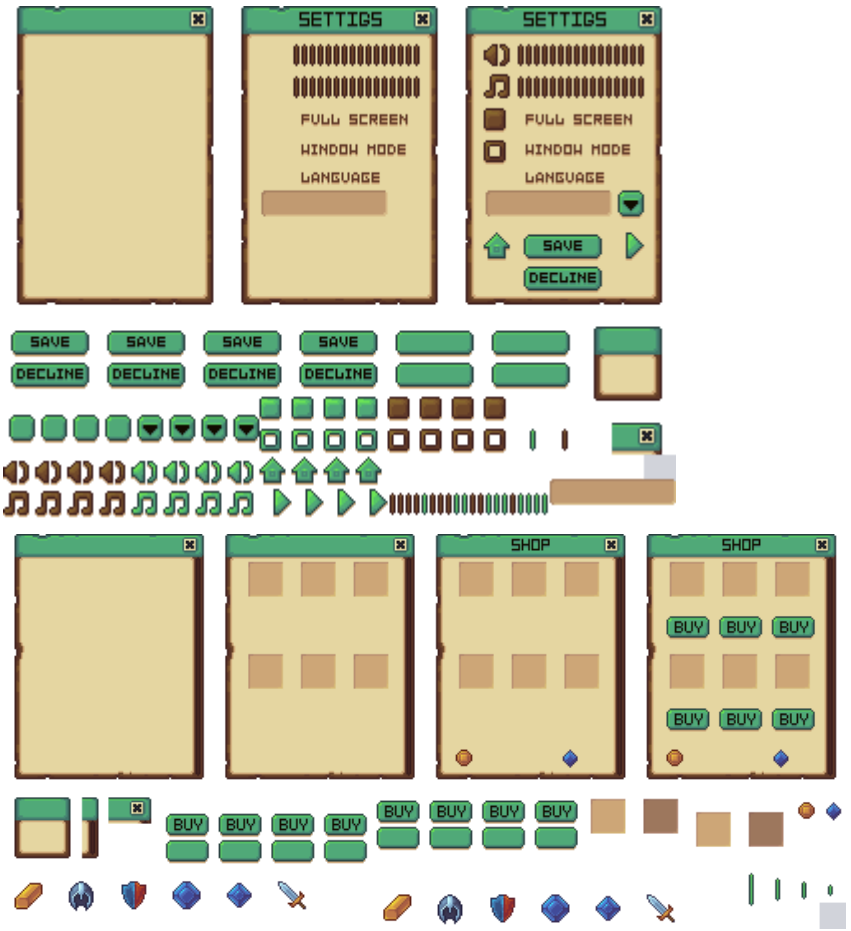
Rouge: réunions
Bleu : soutenances orales
Vert : entraînements pour les oraux

- Site web



- Tile assets




SETTIGS
FULL SCREEN
WINDOW MODE
LANGUAGE
SETTIGS
FULL SCREEN
WINDOW MODE
LANGUAGE
SAVE
DECLINE
SAVE
SAVE
SAVE
SAVE
DECLINE
DECLINE
DECLINE
DECLINE
SHOP
SHOP
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY
BUY