ANNEXE

*Valentin MONGEY*

PROJET D’ISN 2013-2014

THE LEGEND OF THE QUEST

**Cahier des charges**

**The legend of the quest**

1. **Un RPG**

Le but de notre projet est de créer un RPG (role playing game).

Dans un RPG, le principe est de faire évoluer son personnage afin de le rendre plus fort et de pouvoir continuer l’aventure jusqu’à un objectif final. Le joueur atteint cet objectif en réalisant une suite de quêtes qui s’enchaîne dans un ordre précis, c’est-à-dire que le joueur est obligé de finir les quêtes dans cet ordre s’il veut arriver à la dernière quête. Cependant, d’autres quêtes, qualifiées de secondaires et qui sont moins importantes dans l’histoire, permettent de diversifier les objectifs à réaliser et à donner plus de liberté au joueur qui peut alors s’écarter de la trame principale tout en continuant de faire évoluer son personnage.

Au début du jeu, le joueur aura le choix entre 3 types de personnages : paladin, hérétique et assassin, tous ayant des compétences et des attributs propres. La classe aura un rôle important dans l’histoire principale : chaque classe aura sa propre histoire, différente des deux autres. Certains choix à faire seront aussi différents selon la classe, en effet le paladin aura par exemple la possibilité de sauver un inconnu ou de ne rien faire, tandis que l’hérétique aura lui la possibilité de tuer cet inconnu pour récupérer son butin. Le choix de la classe est donc important et permet une rejouabilité importante.

Notre personnage évoluera dans une carte très grande dans laquelle il rencontrera de nombreux combats lui permettant de se rapprocher de l’objectif. Plusieurs évènements aléatoires se déclencheront donnant de la diversité aux déplacements sur la carte.

Dans notre jeu, l’objectif sera d’éradiquer une contamination qui attaque la population locale, pour ce faire notre personnage devra effectuer plusieurs quêtes pour finalement vaincre le boss final responsable de cette contamination. Plus le joueur met de temps à vaincre la contamination, plus celle-ci s’étend sur la carte. Cependant, en même temps que la bataille contre la contamination, le joueur devra faire face à un conflit entre deux des trois races présentent et aura la possibilité de prendre parti dans ce conflit et ainsi d’aider l’une ou l’autre, voire de ne pas participer. Le conflit sera choisi lors la création de la partie.

1. **La carte**

Notre personnage se déplacera sur une carte de très grande taille totalement ouverte (on peut aller partout sur la carte et ce dès le début du jeu). Cette carte sera générée en partie aléatoirement au début de chaque partit. En effet, la carte sera composée de 4 types de continents (désert, plaine, marais et contamination) la forme de ces « continents » est toujours pareil mais la disposition de ceux-ci est aléatoire.

La carte est créée à partir d’images carré mises bout à bout en suivant la disposition dans un fichier texte.

La carte sera parsemée de quelques villages permettant d’accéder à une auberge, un marchand...

Les quêtes auront pour but de faire visiter la carte et de faire avancer le joueur dans son combat contre la contamination mais aussi de lui faire découvrir l’histoire de ce monde ainsi que le pourquoi du comment du conflit en cours.

La carte gèrera également le système de jour/nuit pour nous permettre de n’activer certaines actions que la nuit et ainsi rendre le monde plus dynamique.

1. **Les combats**

Notre personnage sera soumis aléatoirement à des combats. Ces combats devront se déclencher en fonction du « biome » dans lequel se trouve le joueur : en effet, si le joueur se trouve dans la plaine, il ne devra rencontrer que des monstres de la plaine, pareillement pour les autres zones.

Le niveau de ces monstres sera réglé par paliers afin d’éviter que les monstres soit trop puissant par rapport aux joueurs, tout en gardant une part d’aléatoire dans leur force.

Le joueur pourra également être accompagné d’un compagnon qui l’aidera durant ces combats. Ces compagnons auront la possibilité de taper les monstres tout comme le joueur, de se faire taper par les monstres, de mourir mais aussi de se soigner.

A la fin d’un combat, le joueur reçoit une récompense en points d’expérience et en pièces d’or. Les points d’expérience permettent au joueur de monter de niveau et de gagner des points de compétences (trois points de compétences par niveau) qu’il pourra alors répartir comme il le souhaite entre plusieurs caractéristiques.

Le combat prendra en compte les chances de critiques, un coup critique infligeant plus de dégât que l’attaque de base. Cela se fait aléatoirement avec un pourcentage de chance de critique que le joueur peut augmenter grâce aux points de compétences. Il y a aussi la possibilité de rater son attaque avec, comme pour la chance de critique, un pourcentage d’esquive qui pourra aussi être augmenté avec les points de compétences. Les monstres et les compagnons auront aussi la possibilité de faire des coups critique ainsi que d’esquiver les attaques.

Les trois classes seront donc différentes sur ses points, par exemple l’assassin aura plus de chance de coup critique et d’esquiver une attaque, mais fera moins de dégâts « de base » que l’hérétique, mais celui-ci qui tapera plus fort généralement verra ses chances de coup critique et d’esquiver réduite.

1. **Objets/artisanat**

Notre personnage aura un inventaire pour pouvoir stocker différent types d’objets comme des potions, des armes, armures…

Les potions lui permettront d’augmenter temporairement certaines de ses statistiques agissant sur les combats (vie, force, défense…)

Les armures protègeront le joueur, en augmentant une statistique « armure », qui ne peut être augmentée que via les armures équipées.

Les armes augmenteront les dégâts du joueur et pourront influer sur les chances de coup critique.

A la fin de chaque combat, le joueur aura un pourcentage de chance de gagner des objets de constructions qui lui permettront de fabriquer des armes et des armures.

Tous ces objets pourront être vendus chez le marchand pour obtenir de l’or et vider son inventaire.

1. **Auberge**

L’auberge permet en échange d’or de regagner sa vie perdue pendant les combats. Dans l’auberge le joueur peut également faire des « combats d’ivrognes ». Ces combats lui permettent de se battre et de parier sur son combat. L’ordinateur peut également parier aléatoirement sur le joueur ou sur son adversaire virtuel.

Le combat est totalement aléatoire. Si le combat est gagné, le joueur double sa mise et gagne la mise que l’ordinateur a pariée sur lui. De plus sa renommée augmentera de un. Cette renommée rend les parie de l’ordinateur plus important mais réduit aussi les chances de victoire. Si le combat est perdu, le joueur perd toute sa mise et sa renommée descend de un.

1. **Evénements aléatoires**

Aléatoirement sur la carte, des évènements peuvent se déclencher, ceux-ci peuvent être de tous types :

* Rencontre déclenchant un combat.
* Visite d’un personnage voulant devenir votre compagnon.
* Le passage d’un personnage devant vous.
* Evénement entraînant le début d’une quête secondaire.

Une sécurité serra mise en place afin d’éviter qu’un évènement ne se déclenche deux fois de suite.

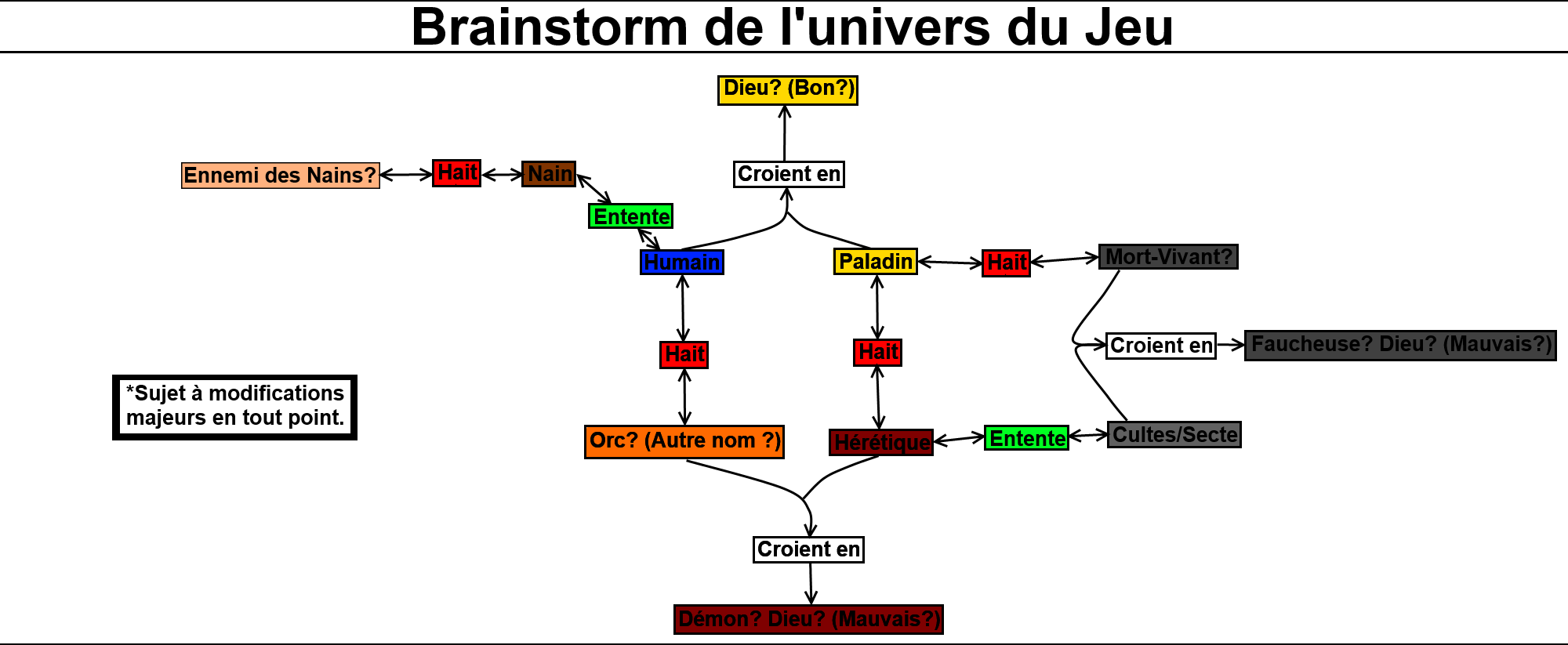
1. **Histoire : conflits internes**

La présence d’une trinité des peuples permet la création de trois conflits distincts. Chaque conflit ayant une origine différente et un déroulement qui lui est propre. Lors de la création de la partie, le conflit est choisi aléatoirement. Le conflit choisi permet de mettre en avant deux des trois races et ainsi permettre au joueur d’en apprendre plus sur ces peuples tout au long de ses aventures. Le conflit n’entre pas directement dans la trame principale qui consiste à combattre la contamination, le joueur n’est pas obligé de prendre part au conflit, cependant il n’est pas à définir comme secondaire car il prend une place importante dans l’histoire et dans le scénario du jeu.

Le joueur peut choisir comment aborder le conflit. Soit il prend parti, soit il ne participe pas. S’il choisit de ne pas prendre parti, le conflit se déroulera sans lui et il ne sera que spectateur. Si au contraire, il choisit de participer, il pourra aider l’un des peuples à prendre le dessus sur l’autre peuple grâce à plusieurs quêtes. Si le conflit tourne en l’avantage du peuple choisi par le joueur, alors celui-ci pourra obtenir des récompenses et des bonus. Mais si le conflit se clos sur une défaite du peuple du joueur, alors celui-ci obtiendra des malus.

Ces bonus et malus auront une influence directe sur la bataille contre la contamination, c’est pour cela que ce conflit ne peut être considéré comme secondaire, car il pourra aider ou désavantager le joueur dans sa quête principale.

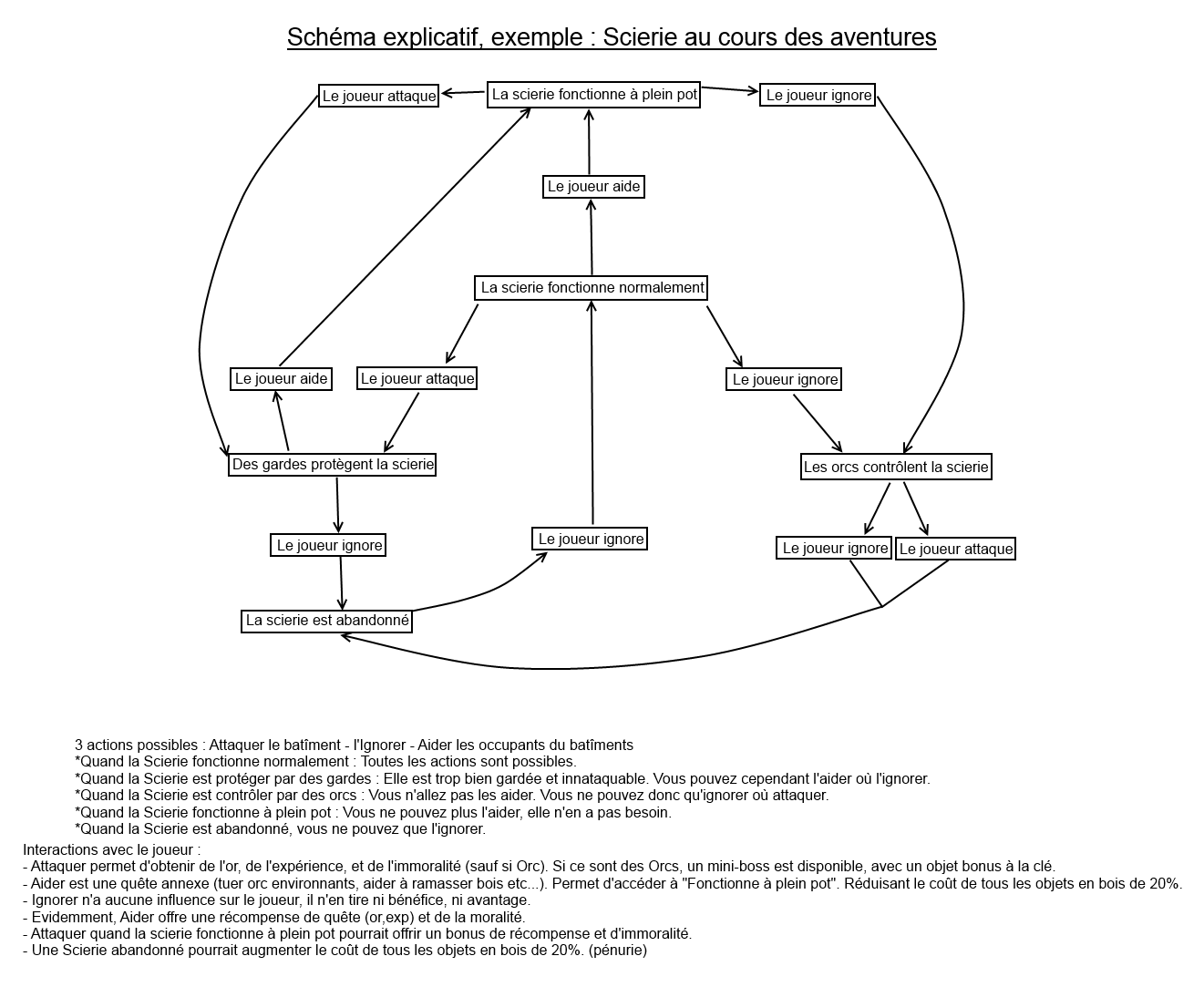
Premières idées sur les relations entre les peuples.



1. New game +

Dans les jeux-vidéos, le new game + se débloque lorsque le joueur a finis le jeu pour la première fois. Il a alors la possibilité de recommencer une partie, cependant il garde des bonus récupérés lors de la partie précédente. Les jeux du style « Rogue » basent leur gameplay autour de ce système, ou le joueur joue un personnage lors de sa partie et récupère par exemple de l’or jusqu’à ce qu’il meurt, définitivement. C’est alors un nouveau personnage, il s’agit dans certains jeux de son descendant (comme dans le jeu Rogue Legacy), et c’est celui-ci qui va dépenser l’or récupérée.

On souhaite intégrer un système de new game +, afin que le joueur puisse, une fois sa quête terminée, recommencer une partie mais que certaines de ses actions soit gravées dans l’histoire.



L’état de la scierie sera conservé lors de la nouvelle partie en new game +.

1. Le programme
   1. Modules

Notre programme étant plutôt gros, nous avons tout fait pour qu’il soit le plus claire possible : nous avons donc fait de nombreux modules, plus ou moins gros, ainsi que plusieurs dossiers. Je vais donc lister les modules en donnant une déscription rapide de leur fonction.

* The Legend of The Quest.py est le fichier principal, il permet de lancer le jeu et renvois les bons « binds » en fonction du titre de la page.

Nous avons ensuite un dossier « module » qui contient tous les autres fichiers python.

* animation.py gère les animations des attaques en combat.
* attaques.py regroupe toutes les attaques du joueur, des compagnons et des monstres.
* auberge\_compagnons.py permet la création de compagnon unique en fonction du biome dans lequel se trouve le village.
* bind.py est le deuxième fichier principal, il permet de lier les touches du clavier à des actions définies.
* combat.py gère tout l’aspect des combats, de la création des ennemis à la victoire ou la défaite du joueur.
* compagnon.py regroupe toutes les caractéristiques de tous les compagnons en fonction de leurs ID.
* ennemi.py fonctionne de la même façon que compagnon.py, regroupe les statistiques des monstres.
* event.py est l’ensemble des évènements aléatoires qui apparaissent sur la carte.
* generation\_map.py permet de générer la carte ainsi que la mini-map en fonction de fichiers textes.
* Item\_ID.py, tout comme compagnon.py et ennemi.py, regroupe les objets du jeu avec leurs caractéristiques et leurs ID.
* marchands.py regroupe toutes fonctions les nécessaires au fonctionnement de la vente, du forgeron, de l’alchimiste.
* menu\_action.py permet simplement d’afficher le menu dans les villages.
* menu\_auberge\_combat.py gère le menu de l’auberge ainsi que la récupération de vie.
* menu\_classe.py permet lors du choix de la classe de créer toutes les statistiques et toutes les « class » du jeu.
* menu\_competence.py est le menu qui affiche les statistiques du joueur, du compagnon et permet de dépenser les points de compétences.
* menu\_esclaves.py gère l’achat d’esclaves en tant que compagnons.
* menu\_inventaire.py affiche l’inventaire, aspect graphique.
* menu\_quete.py est un menu qui affiche les quêtes en cours ou à rendre.
* objet.py permet la gestion des objets et des équipements dans l’inventaire.
* population.py gère la population du jeu et l’avancement de la contamination.
* quete.py recense toutes les quêtes du jeu en attribuant à chacune un ID.
* sauvegarde.py permet la sauvegarde et le chargement des différentes parties.
* statistiques.py est une classe créée uniquement lors de la création de la partie et qui regroupe toutes les caractéristiques du joueur et permet de faire « voyager » de nombreuses variables à travers les modules.
* story.py permet le lancement de la trame principale du jeu et permet la vérification de la victoire ou de l’échec des quêtes.
* worldmap\_v1.py est la gestion de tout ce qui se passe sur la carte comme le lancement aléatoire de combats ou d’évènements.
  1. Ressources

Notre jeu utilise de nombreuses images. Si une image est manquante lors du lancement du jeu, une erreur finira par survenir et le jeu s’arrêtera. Toutes les images sont rangées dans le dossier « img » et sont regroupées en fonction de leur utilité dans le jeu. A la source se trouve toutes les images des menus, du chargement et les images du personnage affichées en combat.

Les dossiers paladin, heretique et assassin regroupent toutes les images qui permettent de donner un effet de mouvement au personnage sur la carte. Il y a huit images par direction.

Dans le dossier Attaque il y a trois dossiers, un par classe, regroupant les images nécessaires aux animations des sorts.

Le dossier Compagnon regroupe les images des compagnons affichées en combat ainsi que les images qui permettent d’animer certains compagnons sur la carte lors d’évènements.

Le dossier event rassemble les images des évènements (autres que ceux des compagnons). Les images sont mises dans des dossiers en fonction de l’évènement auquel elles appartiennent.

Le dossier map contient toutes les images en relation avec la carte (les bloques dont est constitué la carte) comme les différents « biomes », les villages, le HUD mais aussi la carte et la mini-map générées lors de la création de la partie.

Le dossier recense les images qui sont affichées lors du combat de tous les monstres du jeu.

Le dossier objet regroupe les images des objets présents dans le jeu.

Le dossier historique images n’est pas un dossier nécessaire, il ne contient que des images retirées du jeu ou seulement des images montrant les étapes de leur conception.

* 1. Dépendances

Pour pouvoir fonctionner, notre jeu a besoin de :

* Python 3.3
* Pygame 1.3.1
* PILLOW
* Toutes les programmes et ressources cités ci-dessus.

1. Loot

Le loot est géré par une fonction composé en grande partie de if et de elif. Il s’agit en effet de vérifier quel objet va être gagné, soit un objet inutile qui sera tiré aléatoirement dans une liste, soit un objet utile. Dans ces objets utiles, le type de l’objet est choisi entre le bois, le minerai et le tissu. Puis, une vérification du niveau du monstre est faites, car plus le monstre est haut niveau, plus il peut donner des objets de meilleurs qualités.

#Cette fonction permet de gérer le loot sur les monstres avec une grande part d'aléatoire

#De meilleur loot se débloque au fur et à mesure de la montée en niveau des monstres

def lootMonstre(self,personnage,story,item):

if story.activation\_quetes[5] == 1 and self.ID in (301,302,303) and item.inventaireLook(90) < 3:

self.loot = 90

else :

loot = random.randint(1,2) #Soit un loot de type bois/minerais/tissus

#Soit un loot dans la liste lootInutile

if loot == 1:

typeLoot = random.randint(1,3) #Si 1 loot = bois/minerais, si 2/3 loot = tissus

if typeLoot == 1:

choixLoot = random.randint(1,2)#Si 1 loot = minerais, si 2 loot = bois

if choixLoot == 1:

if self.niveau <= 10 :

self.loot = 1

elif self.niveau > 10 and self.niveau <= 15:

qualiteLoot = random.randint(1,2)

if qualiteLoot == 1 :

self.loot = 1

elif qualiteLoot == 2 :

self.loot = 2

elif self.niveau > 15:

qualiteLoot = random.randint(1,3)

if qualiteLoot == 1 :

self.loot = 1

elif qualiteLoot == 2 :

self.loot = 2

elif qualiteLoot == 3 :

self.loot = 3

elif choixLoot == 2:

if self.niveau <= 10 :

self.loot = 4

elif self.niveau > 10 and self.niveau <= 15:

qualiteLoot = random.randint(1,2)

if qualiteLoot == 1 :

self.loot = 4

elif qualiteLoot == 2 :

self.loot = 5

elif self.niveau > 15:

qualiteLoot = random.randint(1,3)

if qualiteLoot == 1 :

self.loot = 4

elif qualiteLoot == 2 :

self.loot = 5

elif qualiteLoot == 3 :

self.loot = 6

else:

if self.niveau <= 10 :

self.loot = 7

elif self.niveau > 10 and self.niveau <= 15:

qualiteLoot = random.randint(1,2)

if qualiteLoot == 1 :

self.loot = 7

elif qualiteLoot == 2 :

self.loot = 8

elif self.niveau > 15:

qualiteLoot = random.randint(1,3)

if qualiteLoot == 1 :

self.loot = 7

elif qualiteLoot == 2 :

self.loot = 8

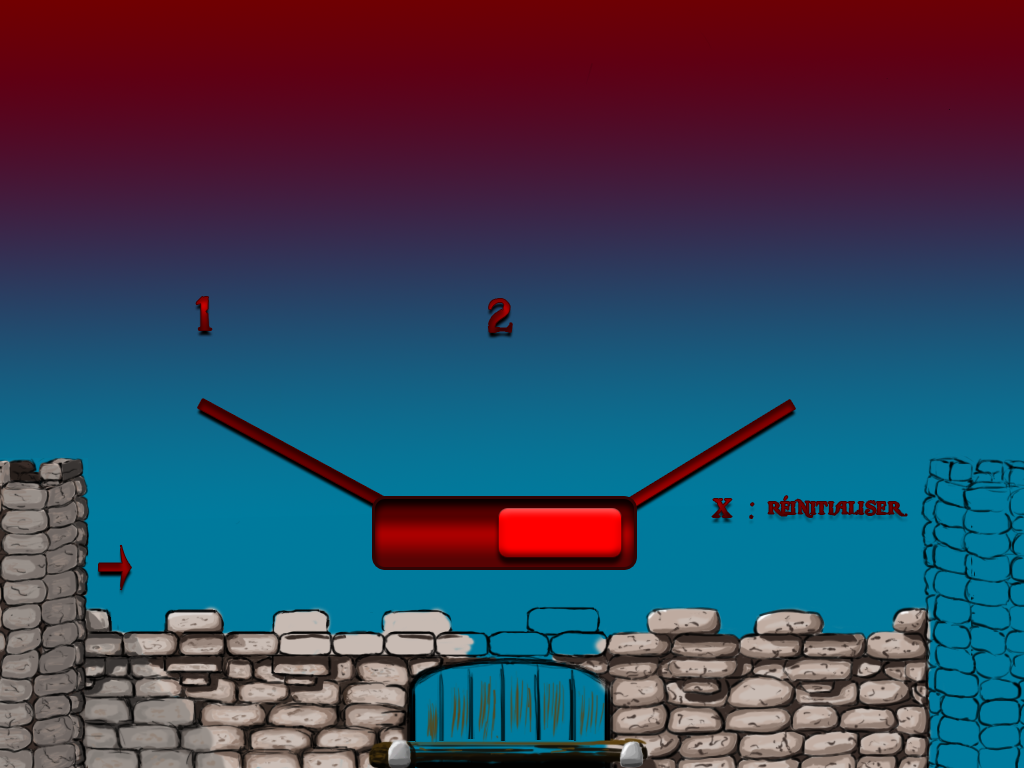
elif qualiteLoot == 3 :

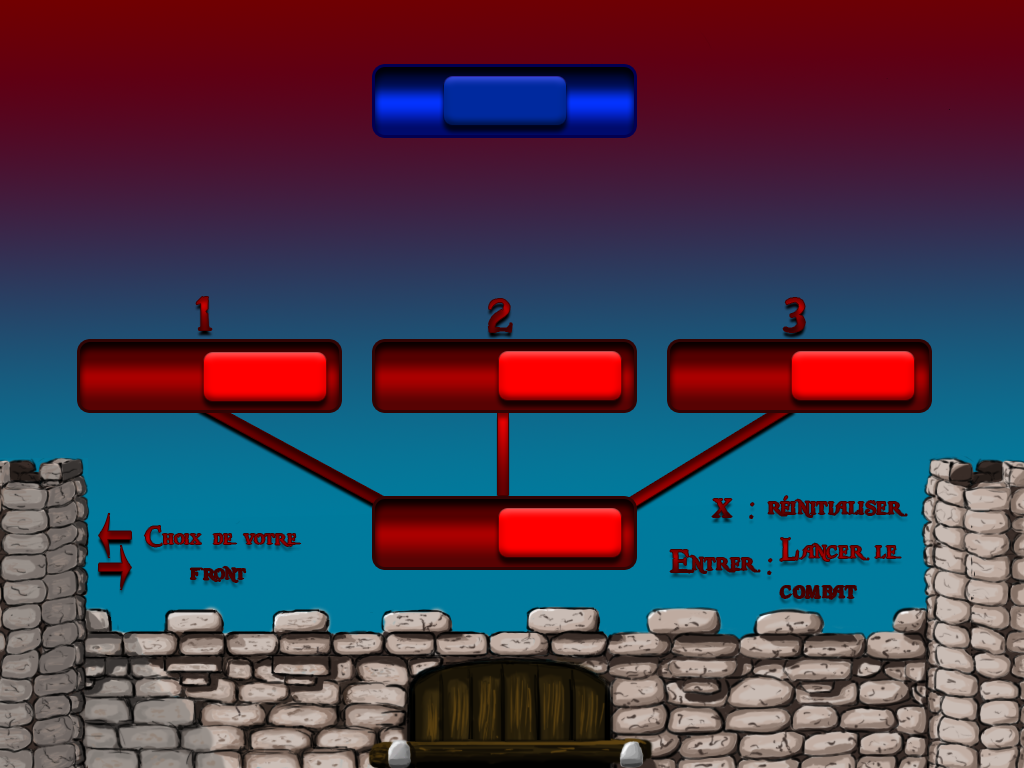
self.loot = 9

elif loot == 2:

self.loot = random.choice(item.lootSecondaire)

1. Images

Menu de la bataille finale dessiné de A à Z par Valentin.



Création du compagnon scientifique utilisé lors d’une quête.



Création d’un monstre utilisé dans la zone de la plaine.

1. Attaque

Cette fonction est l’attaque du monstre numéro 1 : elle fait toute les vérifications (comme la présence ou non d’un compagnon) et inflige des dégâts en fonction des dégâts minimum et maximum du monstre.

def attaqueMonstre1(combat,fen1,personnage,item,comp):

if combat.stunMonstre1 == 0 :

toucherMonstre1 = random.randint(1,100) #Chance de toucher

dgtMonstre1 = random.randint(combat.monstre1.dgtMin,combat.monstre1.dgtMax) #Degat aléatoire entre les dgts min et max

#Si l'ID du monstre est dans la liste des monstres de type HEAL, choix entre attaque (1 et 2) et soin (3)

if combat.monstre1.ID in combat.monstre1.listHeal :

choixMonstreAttaque = random.randint(1,3)

#Si l'ID du monstre n'est pas dans la listre des monstre de type HEAL, ou le choix est 1 ou 2

if combat.monstre1.ID not in combat.monstre1.listHeal or choixMonstreAttaque < 3 :

if comp.ami !=0 :

#Vérifie si le compagnon est de type tank ou non, si oui nombre aléatoire, si 1 attaque le joueur, sinon attaque le compagnon

if comp.IDCompagnon not in comp.listTank : choixAttaque = random.randint(1,2)

else : choixAttaque = random.randint(1,4)

#Si le joueur n'a pas de compagnon, ou le choix de l'attaque est un, ou le compagnon est mort

if comp.ami == 0 or choixAttaque == 1 or comp.vie <= 0:

dgtMstr1 = dgtMonstre1 - (personnage.constitution + personnage.armure) #Degat du monstre (dgtMonstre1) - la resistance du joueur

if toucherMonstre1 > personnage.esquive : #Le monstre touche le joueur

if dgtMonstre1 <= personnage.constitution + personnage.armure : #Degats du monstre inférieur à la resistance du joueur

dgtMstr1 = random.randint(0,1) #1 chance sur 2 de faire soit 1 soit 0 de degat au joueur

if dgtMstr1 == 1 : personnage.vie -= 1

else : personnage.vie -= 0

else :

personnage.vie -= dgtMstr1 #Degat du monstre - resistance du joueur

textDegatMonstre1 = combat.fondtext.render("-" + str(dgtMstr1),1,(255,0,0))

else : textDegatMonstre1 = combat.fondtext.render("Rate",1,(0,0,255))

if comp.ami == 0 : fen1.blit(textDegatMonstre1,(210,270))

else : fen1.blit(textDegatMonstre1,(210,330))

else :

dgtMstr1 = dgtMonstre1

if toucherMonstre1 > comp.esquive :

comp.vie -= dgtMstr1

textDegatMonstre1 = combat.fondtext.render("-" + str(dgtMstr1),1,(255,0,0))

else : textDegatMonstre1 = combat.fondtext.render("Rate",1,(0,0,255))

fen1.blit(textDegatMonstre1,(210,155))

else :

choixMonstre\_a\_Heal(combat,fen1,personnage,item,comp)

if choixMonstreHeal == 4 : attaqueMonstre1(combat,fen1,personnage,item,comp)

else : monstreHeal1(combat,fen1,personnage,item,comp)

1. Victoire

Cette fonction est appelée à chaque tour en combat pour vérifier si les conditions de victoire sont réunies. Si elles le sont, alors plusieurs modifications sont apportées au joueur, comme le gain d’expérience, s’il passe un nouveau palier d’expérience et donc passe de niveau, s’il gagne des objets, etc.

#Fonction vérifiant si les conditions pour la victoire sont réunis et permet de gérer le gain d'experience

#ainsi que le passage de niveau lorsque l'exp est superieur à l'exp max du niveau.

def victoire(self,fen1,personnage,item,comp,story):

#Si tout les monstres meurent, victoire du personnage

if self.monstreVivant1 == 0 and self.monstreVivant2 == 0 and self.monstreVivant3 == 0 :

#Reduit le temps de recharge du marchand d'esclave

if personnage.vendeurEsclave != 0 : personnage.vendeurEsclave -= 1

#Supprime le compagnon s'il est mort

if comp.ami != 0 and comp.vie <= 0 : comp.ami = 0

time.sleep(0.8)

pygame.display.set\_caption("Victoire")

fen1.blit(self.fond, (0,0))

textVictoire = self.fondtext.render("Victoire",1,(0,0,0))

fen1.blit(textVictoire,(470,160))

#Gain d'experience à la mort du monstre si 1 monstre

if self.nombreMonstre == 1 :

gainXp = self.monstre1.gainxp

gainOr = self.monstre1.gold

#Gain d'experience à la mort du monstre si 2 monstres

if self.nombreMonstre == 2 :

gainXp = self.monstre1.gainxp + self.monstre2.gainxp

gainOr = self.monstre1.gold + self.monstre2.gold

#Gain d'experience à la mort du monstre si 3 monstres

if self.nombreMonstre == 3 :

gainXp = self.monstre1.gainxp + self.monstre2.gainxp + self.monstre3.gainxp

gainOr = self.monstre1.gold + self.monstre2.gold + self.monstre3.gold

personnage.xp += gainXp

personnage.argent += gainOr

#Passage d'un niveau quand l'xp dépasse l'xpmax, gain de points de compétences, régénération de la vie au max, augmentation de l'xpmax à atteindre

if personnage.xp >= personnage.xpmax :

personnage.up = 1

personnage.niveau += 1

personnage.competence += 5

personnage.vie = personnage.viemax

personnage.xpmax += personnage.xpbase + (personnage.niveau - 1) \* 250

textPassageNiveau = self.fondtext.render("Vous avez gagné un niveau !",1,(0,0,0))

#Passage de niveau lors de la quête 7

if story.activation\_quetes[7] != 1 :

fen1.blit(textPassageNiveau,(300,260))

#Appel de la fonction loot qui permet de gagner des objets à la fin du combat

self.lootFinCombat(fen1,personnage,item,comp,story)

textGainXp = self.fondtext.render(str(gainXp) + " points d'expérience gagnés.",1,(0,0,0))

if gainOr > 1 : textGainOr = self.fondtext.render(str(gainOr) + " pièces d'or récupérées.",1,(0,0,0))

else : textGainOr = self.fondtext.render(str(gainOr) + " pièce d'or récupérée.",1,(0,0,0))

#Si la quête de la bataille finale n'est pas activé

if story.activation\_quetes[7] != 1 :

#Si l'inventaire n'est pas plein, retour sur la carte

if self.inventairePlein == 0 :

textRetourMap = self.fondtext.render("1.Carte",1,(0,0,0))

fen1.blit(textRetourMap,(300,440))

#Si l'inventaire est plein, le joueur peut ouvrir son inventaire et supprimer des objets, ou abandonner les objets gagnés

else :

textInventaire = self.fondtext.render("1.Ouvrir l'inventaire.",1,(0,0,0))

textRetourMap = self.fondtext.render("2.Abandonner les objets.",1,(0,0,0))

fen1.blit(textInventaire,(300,440))

fen1.blit(textRetourMap,(500,440))

#Si la quête de la bataille finales est activé, la victoire est particuliere, le joueur doit recevoir un rapport spécial

#mais ne doit pas retourner sur la carte.

if story.activation\_quetes[7] == 1:

fen1.blit(self.fond, (0,0))

textVictoire = self.fondtext.render("Vous avez gagné la bataille, vous recevez un rapport des autres fronts.",1,(0,0,0))

textCharger = self.fondtext.render("1.Voir le rapport.",1,(0,0,0))

fen1.blit(textVictoire,(300,200))

fen1.blit(textCharger,(300,240))

if story.activation\_quetes[7] != 1 :

fen1.blit(textGainXp,(300,210))

fen1.blit(textGainOr,(300,310))