

Table des matières

1	Introduction.....	2
2	Cahier des charges.....	2
2.1	Présentation.....	2
2.2	Objectif.....	2
2.3	Scénario.....	2
2.4	Déroulement du jeu.....	2
2.5	Règles.....	3
2.6	IA.....	3
2.7	Détails techniques.....	3
2.8	Public visé.....	4
2.9	Supports logiciels utilisés.....	4
2.10	Interlocuteurs.....	5
2.10.1	Promoteurs.....	5
2.10.2	Référents.....	5
2.11	Ressources apportées.....	5
2.11.1	Support informatique.....	5
2.11.2	Support papier.....	6
2.12	Spécifications techniques.....	6
2.12.1	Tâches effectuées par le programme.....	6
2.12.2	Tâches non effectuées par le programme.....	6
2.13	Structure du jeu.....	8
2.14	Ébauche graphique.....	8
2.14.1	Écran principal.....	8
2.14.2	Écran jeu.....	9
2.14.3	Écran Campagne.....	9
2.14.4	Écran Option.....	10
2.14.5	Écran de jeu.....	10
2.15	Bases de données.....	11
2.16	Prévision des cas de test.....	11
3	Description des tâches.....	12
4	Algorithmes, UML et Base de données.....	12
4.1	Analyse des algorithmes.....	12
4.1.1	L'IA.....	12
4.1.2	La boucle de jeu.....	12
4.1.3	Schéma UML.....	13
4.1.4	Schéma Tkinter.....	13
4.2	Base de donnée.....	13
4.2.1	Schéma conceptuel.....	13
4.2.2	Schéma relationnel.....	13
4.2.3	Contraintes d'intégrité.....	13
5	Mode d'emploi du programme.....	13

1 Introduction

Le travail de fin d'année, est un exercice qui mets nos compétences à rude épreuve. Il nous permet d'apprendre à gérer notre temps, à avoir des objectifs clairs. Étant resté sur ma faim avec le travail de fin d'année de l'année dernière, j'avais envie de me lancer sur un gros projet. Je sais depuis le départ que je n'arriverais sans doute pas à faire tout ce que je voulais, mais me voilà avec mon jeu original, création de mon imagination.

2 Cahier des charges

2.1 Présentation

Il s'agit d'un jeu de gestion et d'évolution de ville dans l'ère médiévale mettant l'accent sur la gestion de ressources limitées dans le but de prospérer. L'idée est de représenter au mieux le mode de vie et les liens entre les différentes personnalités actives dans une ville, ainsi que les différents conflits à travers un scénario fictif dans un monde alternatif de mises en situation.

2.2 Objectif

L'objectif premier est de terminer le scénario résumé ci-dessous. Mais un ou plusieurs objectifs seront donnés pour la « mission » en cours (à savoir la partie d'histoire jouée sur le moment). Des objectifs simples tels qu'amasser moult richesses, atteindre un nombre donné de population ou encore vendre une partie des ressources amassées dans un temps limite (Ceci est cité à titre d'exemple et pourra être présent sous d'autres formes dans le jeu). Une fois le ou les objectifs atteint(s), la mission prend fin et débloque la mission suivante. L'impossibilité de remplir le ou les objectifs, si cela est possible, n'entraînera aucune conséquence à part le recommencement de la mission.

2.3 Scénario

Étudiant récemment diplômé de la prestigieuse université d'Eona en gestion et commerce, la capitale commerciale du grand empire Aturéen, vous essayez de vous faire une place dans ce monde. Malheureusement l'empire est en guerre contre les puissants guerriers du Modég. Mais bien pire est arrivé, la famine fait des ravages et a mis l'économie à terre. Une défaite face au Modég semble se profiler à l'horizon. Le roi a besoin de vous pour rétablir la situation, pour amener l'empire à être prospère. Serez-vous à la hauteur ? Saurez-vous faire face à l'envahisseur ?

2.4 Déroulement du jeu

Le jeu en mission se déroule dans un environnement graphique 2D avec la caméra à la verticale du sol et arrive sur une carte avec quelques bâtiments construits à l'avance symbolisant le village que le joueur va gérer. Le joueur va alors commencer à prendre des décisions au fil du temps qui passe

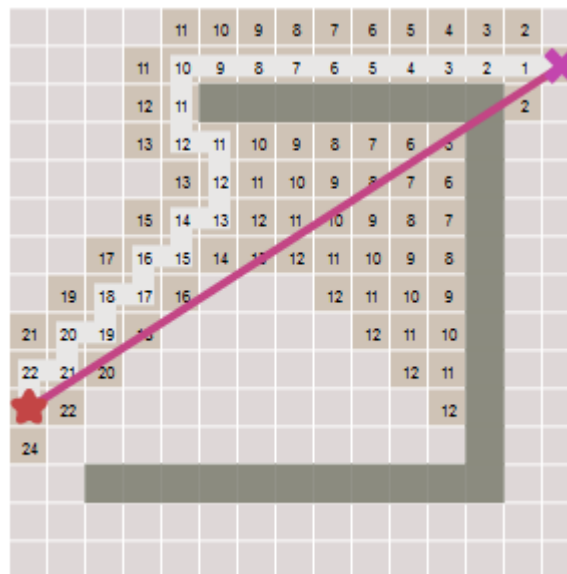
comme construire un nouveau bâtiment ou récolter des ressources dans le but, ou pas, de remplir son objectif.

2.5 Règles

Le jeu en mission est régi par le temps qui passe (en jours). Il est limité par les bords de la carte ainsi que les obstacles naturels tel que des arbres ou montagnes et les bâtiments. Les ressources présentes sur la carte sont en quantité limitée dans un premier temps mais pourront être produites ou substitués par d'autre ressource plus évoluée. La croissance de la population se fera en fonction de l'abondance de nourriture, des habitations disponible mais aussi de la richesse du village. Le village pourra aussi subir des attaques aléatoire mais un village sans défense aura plus de chance de se faire attaquer qu'un village fortifié et gardé (mais subira des attaques plus importantes).

2.6 IA

L'IA de déplacement des entités (tous les personnages ou animaux qui sont mobile sur le plateau de jeu) sera calculée grâce à un vecteur entre le point de départ et le point d'arrivée pour déterminer la direction de la recherche de chemin dans le but de ne pas devoir calculer toute la carte à chaque déplacement, ensuite les cases entre le point de départ et le point d'arrivée seront numérotées grâce à un étalement en croix car aucun déplacement en diagonale n'est possible, enfin les numéros seront additionnés et le plus court, sélectionné.



2.7 Détails techniques

La carte se découpe en quadrillage sur deux plans : le sol qui détermine la vitesse de déplacement et le niveau obstacle qui prendra en compte tous les bâtiments et obstacles naturels pour le déplacement des personnages. Les villageois sont produits lors qu'il existe un surplus d'habitation par rapport aux nombres de villageois déjà présent. Chaque villageois pourras être affecté a un métier qui sera déterminé par le bâtiment auquel le joueur l'aura affecté, donc a la base, un villageois est « vierge », il n'a aucune fonction dans le village. Envoyer un villageois dans une mine en fera un mineur, un autre dans un champ, un fermier ou encore a la caserne, un soldat.

Une fois le villageois affecter, il commencera à travailler à l'endroit affecter par son bâtiment base. Pour les emplacements de travail étant loin du bâtiment de base, ce qui arrive pour les mines et carrières, le villageois apportera le fruit de son dur labeur à son bâtiment base lors que son inventaire sera remplis. Pour ceux dont le travaille se passe à l'intérieur du bâtiment base, l'inventaire s'incrémentera au fil du temps en fonction du nombre de personnes employées. Mais l'aspect le plus important sera le transfert de matériaux d'un bâtiment à un autre, car certains ont besoin d'objet produit ailleurs pour eux-mêmes produire, il faudra donc mettre des villageois dans un entrepôt qui s'occuperont de transporter les ressource du bâtiment producteur jusqu'à l'entrepôt puis de ce dernier vers un bâtiment en besoin.

Chacun villageois acquerra une certaine expérience dans le domaine dans lequel il travaille, dont plus y travaillera, plus la production sera rapide, changer un villageois de métier lui fera perdre l'expérience de son précédent métier petit à petit.

Le joueur pourra construire des routes pour accélérer la vitesse des villageois qui sera donc pris en compte avec l'IA de déplacement qui utilisera en priorité les routes, si aucune route pour la destination n'est trouvée, l'IA utilisera simplement le chemin le plus court. Si un chemin existe mais qu'il est jugé trop long par l'IA, le villageois pourra alors ne pas utiliser le chemin existant et passera alors par son algorithme pour trouver le chemin le plus rapide.

2.8 Public visé

Ce jeu est orienté pour des joueurs calmes de tout âge pouvant utiliser un ordinateur dans le but de se détendre sans être stressé par le jeu en question. Commencer l'aventure vidéo-ludique avec ce jeu est tout à fait possible car les mécanismes de base et leurs utilisations sont simples mais permettent avec le temps d'arriver à un niveau de complexité relativement important.

2.9 Supports logiciels utilisés

Pour la réalisation de ce projet plusieurs programmes seront utilisés dans le cadre de la création de schémas et divers contenus, en voici la liste et une brève description :

- Audacity / Reaper
Création de contenus audio
- Inkscape
Création de dessin vectoriel
- Gimp
Création et modification d'image
- Libre Office
Divers outils de bureautique classique (Word, Excel, Powerpoint, ...)
- Scribus
Mise en page de document texte
- Visual Studio Code, Atom, Notepad++
Éditeur de code
- Github
Site d'hébergement et de gestion de développement de logiciels
- Trello

- Outil de gestion de projet (Todo list)
- ArgoUML
Logiciel de création de schémas
- SqliteSpy
Outil de gestion de bases de données
- pygame
moteur de jeu en python

2.10 Interlocuteurs

2.10.1 Promoteurs

- M^{me} Eloy – Laboratoire Logique/Python – isabelleeloy@ipet.be
- M^{me} Boulogne – Laboratoire Informatique/Informatique – isabelleboulogne@ipet.be

2.10.2 Référents

- M. Dewit Alexandre – Programmeur – alexandredewit97@gmail.com

J'ai 18 ans et je suis actuellement étudiant dans le domaine des sciences informatiques. Ce dont j'adore passer mon temps : le sport, le développement de site internet, de programmes et les jeux vidéo.

- M. Elkinoo – Joueur – ElkinooTV@gmail.com

Je me nomme Elkinoo, j'ai décidé de partager ma passion et mes expériences de gaming. Je ne suis pas un joueur pro et je n'aspire pas à l'être. Je ne suis pas spécialement une personne qui ne joue que pour le défi même si j'aime bien me torturer l'esprit et la patience sur certains jeux. Je ne souhaite pas non plus absolument terminer tous les jeux que je touche. Par contre, je suis du genre à vouloir tout essayer, et avec passion... Parce que ce que j'aime, ce sont les jeux-vidéos, en général.

2.11 Ressources apportées

2.11.1 Support informatique

- www.developpez.com/ - Site / Forum de programmation
- rimworldgame.com/index.php?lang=fr – Site / Jeux, inspiration
- lethispop.com/index.php?lang=fr – Site / Jeux, inspiration
- www.ageofempires.com/ - Site / Jeux, inspiration
- www.jeuxvideo.com/jeux/pc/00003654-empire-earth.htm – Site / Jeux, inspiration
- <https://qiao.github.io/PathFinding.js/visual/> - Visualisation d'algorithme pour du pathfinding

- <http://www.redblobgames.com/pathfinding/a-star/introduction.html> - Visualisation d'algorithme pour du pathfinding
- <http://python.cocos2d.org/> - module python
- <http://superpowers-html5.com/index.fr.html> – pack pour texture

2.11.2 Support papier

- CHAZALLET Sébastien, *Python 3 : Les fondamentaux du langage*, France, ENI, 2014, RessourcesInformatique
- EBEL Franck, Rohaut Sébastien, *Algorithmique : Techniques fondamentales de programmation*, France, ENI, 2014, RessourcesInformatique
- ACISSI, *Sécurité informatique : Ethical Hacking (Apprendre l'attaque pour mieux se défendre)*, France, ENI, 2015, Informatique Technique
- ROBERT Dimitre, *Gimp 2.8 : Débuter en retouche photo et graphisme libre*, Paris, Eyrolle, 2014, Accès Libre
- GNU/Linux Magazine, *Python : Le guide pour devenir un véritable expert du langage !*, France, Les Éditions Diamond, Hors-Série N°73
- GNU/Linux Magazine, *Analyse de donnée & big data : Le guide pour manipuler et analyser vos données efficacement !*, France, Les Éditions Diamond, Hors-Série N°78
- LinuxUser & Developer, *The Python Book*, UK, Imagine Publishing
- Patrois N. (2016). Parcourir des graphes en largeur. GNU/Linux Magazine, 189, 22-31

2.12 Spécifications techniques

2.12.1 Tâches effectuées par le programme

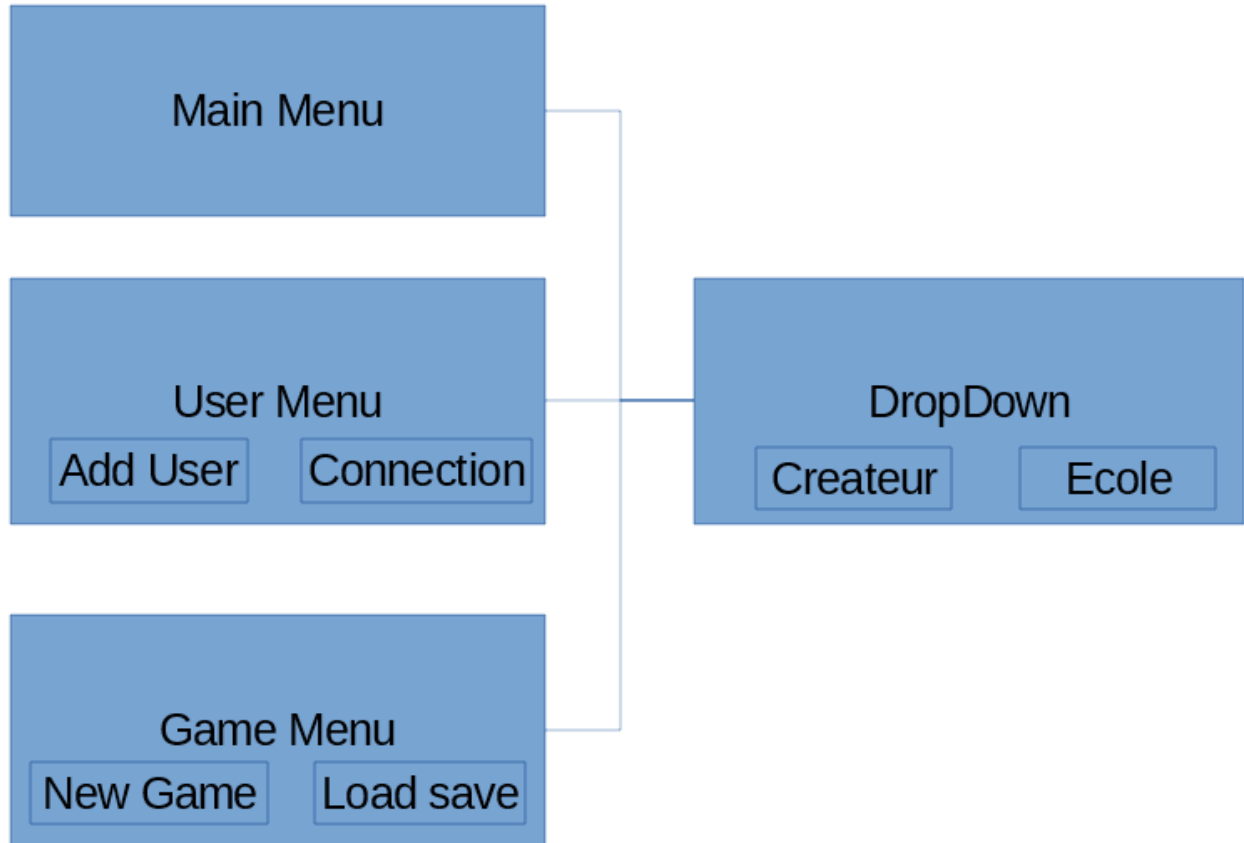
- Gestion de différents profils d'utilisateur pour séparer les sauvegardes
- Charger et effectuer des sauvegardes
- Affichage des ressources possédées par le joueur mise à jour en temps réel
- Affichage des différentes ressources nécessaires au bon fonctionnement d'un bâtiment
- Déplacement des personnages grâce à une IA
- Gestion du temps

2.12.2 Tâches non effectuées par le programme

- Cycle jour/nuit
- Mode de jeu libre (hors scénario)
- Sauvegarde automatique ponctuel
- Affichage du chemin pris par une entité appartenant au joueur lors de la sélection de celle-ci

- Bâtiment multi-structurel (ou fusion d'un ou plusieurs bâtiments de même type pour accroître sa fonctionnalité en dépit d'un prix de construction plus élevé)
- Gestion d'une carte

2.13 Structure du jeu



2.14 Ébauche graphique

Fichier	Action	A Propos	
---------	--------	----------	--

Single Player

Game Settings

Exit Game

2.14.1 Écran principal

Première fenêtre du jeu permettant d'accéder au jeu, aux options ou encore sortir du jeu.

2.14.2 Écran jeu

Fichier	Action	A Propos	
---------	--------	----------	--

Playeur:

Campaign

Main Menu

Écran de sélection du type de partie qui se limite actuellement a une campagne scénarisée. Un champ joueur permet d'entrer un nom qui va caractériser une sauvegarde, une liste des joueurs déjà enregistrer peu être envisagé.

2.14.3 Écran Campagne

Écran permettant le démarrage d'une nouvelle partie, chaque nouvelle partie écrase la sauvegarde

Fichier	Action	A Propos	
---------	--------	----------	--

New Game

Load

Game Menu

précédente si elle existe. Il est possible aussi de charger une partie, qui sera directement chargé en fonction du nom de joueur spécifier dans l'écran de jeu. Ce système de sauvegarde permet d'éviter la triche sur des événements aléatoires en sauvegardant puis rechargent.

2.14.4 Écran Option

Fichier	Action	A Propos	
---------	--------	----------	--

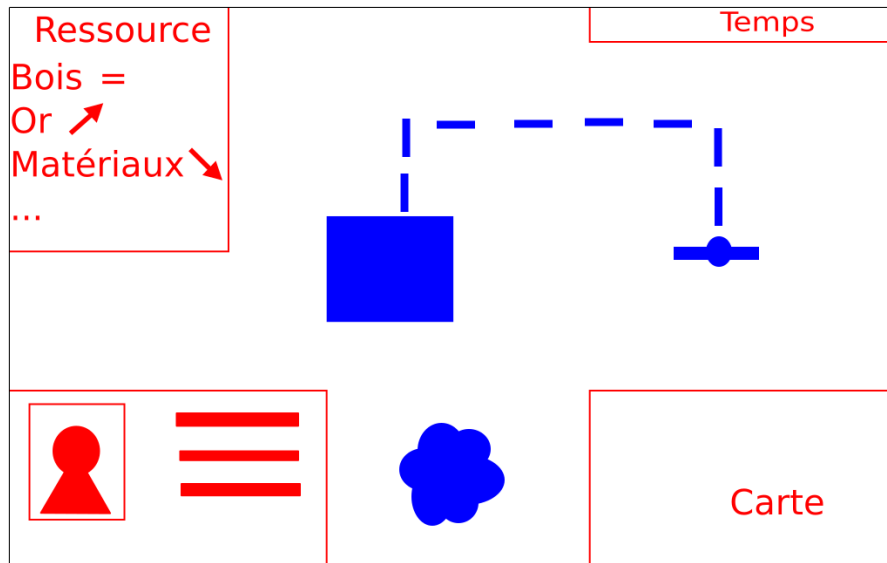
Volume :

DefaultCancel

OK

Écran de choix des options, avec le reset des options de départ, l'annulation qui revient sur l'écran principal sans garder en mémoire les changements effectué, ainsi qu'un bouton de validation appliquant les modifications.

2.14.5 Écran de jeu



L'écran de jeu est divisé en trois calques qui sont ici représentés par les différentes couleurs du plan le plus éloigné au plus proche (Les couleurs de l'écran de jeu sont seulement représentatives des différents calques et ne sont donc pas les couleurs finales du jeu).

Noir – Background : C'est le fond du jeu sur lequel se trouvera la map représentative de la terre, donc avec l'herbe, les roches, ... Certains obstacles seront donc infranchissables de par la nature du terrain.

- **Bleu :** Cette partie du plan est consacrée à tout ce qui va changer au fil du jeu, les constructions, les entités qui se déplacent, les différentes ressources exploitables, ainsi que la représentation de la sélection d'une entité ou d'un bâtiment.
- **Rouge – Interface utilisateur :** Cette partie va fournir au joueur toutes les informations que le joueur pourrait avoir besoin lors de sa partie, dont les différentes parties sont décrites ci-dessous
 - **Ressource :** Cette partie listera toutes les ressources possédées par le joueur, s'il en produit ou s'il est en perte.
 - **Information sélection :** Cette partie, située en bas à gauche, affichera toutes les informations disponibles pour la sélection en cours, que cela soit une entité ou un bâtiment ainsi que les actions possibles directement.
 - **Carte :** Cette partie sera une représentation miniature de tout l'espace jouable.
 - **Temps :** Cette partie sera consacrée à l'écoulement du temps. L'incrément d'une date

À savoir que les deux derniers points cités seront codés seulement si mes capacités me le permettent comme indiqué précédemment.

2.15 Bases de données

- Le level en cour
- Nom et position des batiment
- Nom et position des entité
- Nom et postition des ressource
- Utilisateur et mot de passe
- Nom de sauvegarde

2.16 Prévision des cas de test

- Test de l'IA de déplacement
 - Collision avec les obstacle
 - Cohérence du chemin choisi par l'IA
- Production de villageois au fil du temps
- Test de superposition des niveaux lors de création de structure
- Sauvegarde complète et correcte

3 Description des tâches

La première grosse partie a été l'apprentissage et la compréhension de pygame qui n'a pas été sans mal puisque la documentation de pygame est loin d'être complète et précise.

La deuxième partie, c'est toute la partie jeu. Je me suis préoccupé d'avoir un jeu qui fonctionne sans avoir vraiment une interface graphique

Et enfin, la construction des fenêtres, de la base de données et quelque petit détail sur le jeu.

4 Algorithmes, UML et Base de données

4.1 Analyse des algorithmes

4.1.1 L'IA

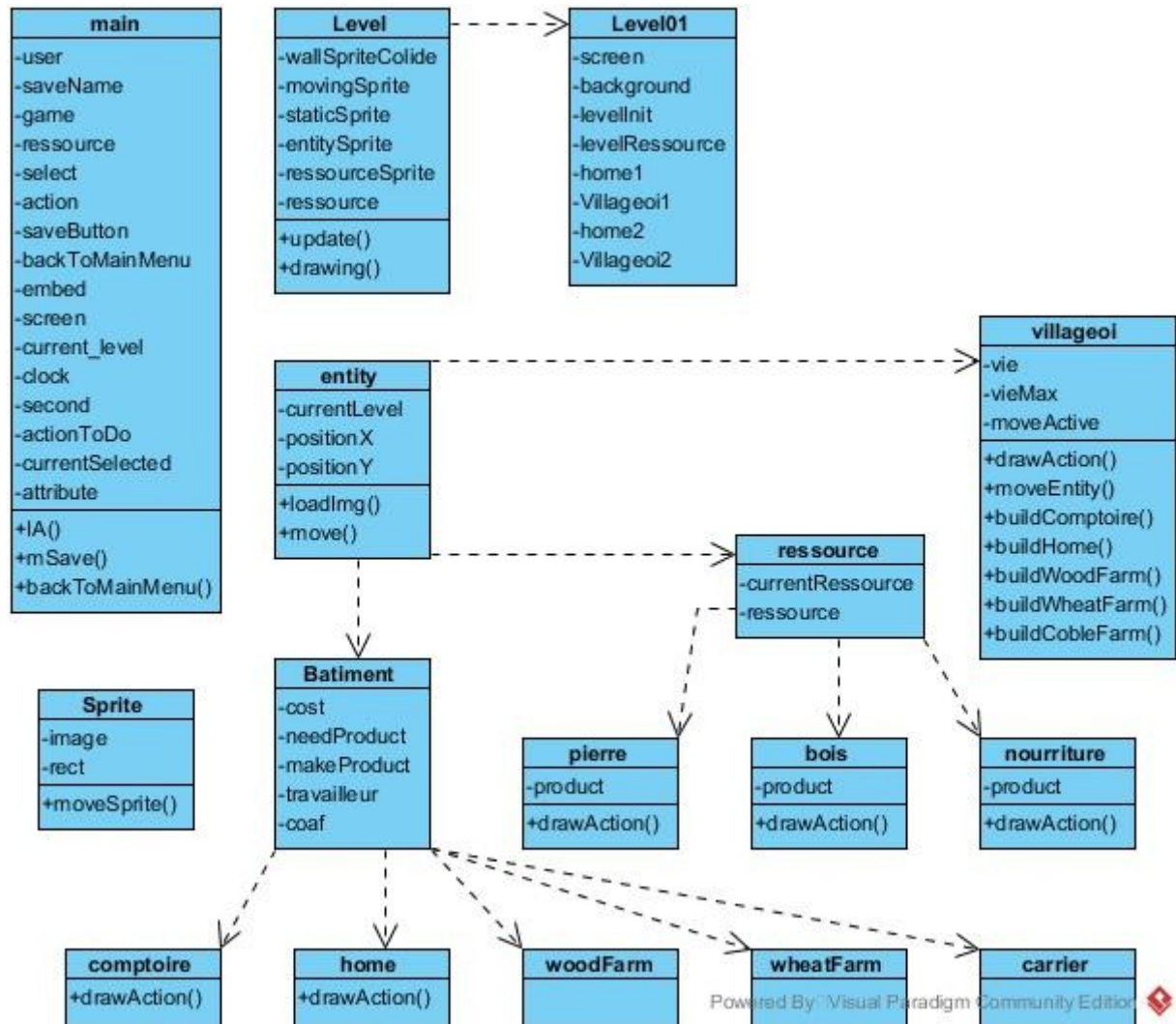
Pour chaque calcul de chemin, je commence par rechercher tous les points où se trouve un obstacle, puis avec ces données je simule la création d'une grille avec laquelle l'IA va travailler. L'IA numérote la grille en se dirigeant vers le point d'arrivée, quand il atteint le point d'arrivée, il s'arrête. Il ne reste plus qu'à remonter les numéros jusqu'au point de départ.

4.1.2 La boucle de jeu

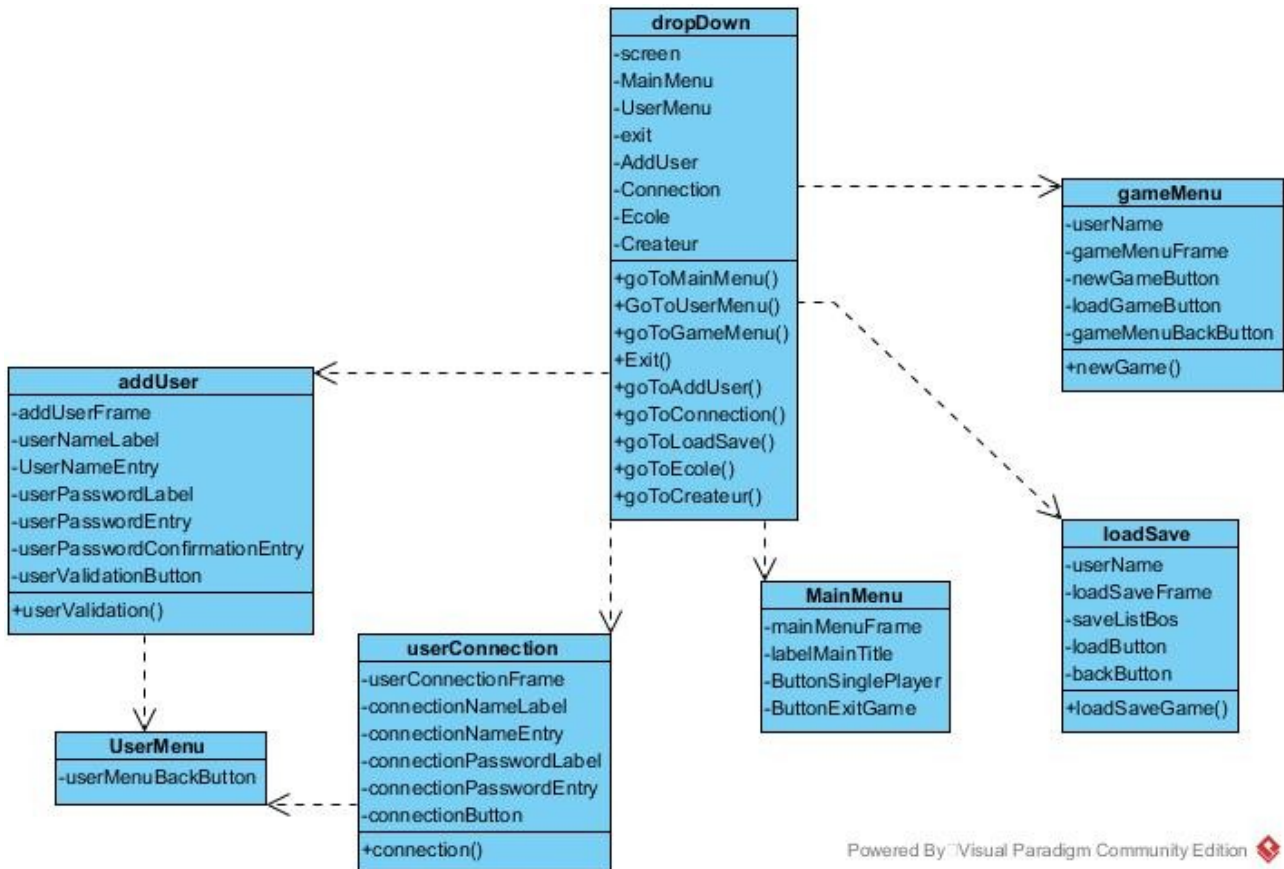
La boucle de jeu est ce qui permet au jeu de récupérer et d'effectuer toutes les actions ordonnées par l'utilisateur et de mettre à jour l'interface. Elle se déroule comme suit :

1. Mise à jour de l'affichage des ressources
2. Récupération des événements entrés par l'utilisateur
3. Si l'utilisateur a un villageois sélectionné et qu'il clique droit sur la carte, il déplace l'entité sélectionnée.
4. Si l'utilisateur fait un clic gauche sur un élément de la carte, il affiche ses informations et action possible.
5. Toutes les 300ms, déplacement des entités qui doivent se déplacer
6. Toutes les 500ms, action des entités sur les ressources de la carte.
7. Toutes les secondes, consommation et/ou la production de ressources par les bâtiments concernés.
8. Gestion des objets en cours de placement
9. Mise à jour et affichage de tous les éléments.

4.1.3 Schéma UML



4.1.4 Schéma Tkinter



4.2 Base de donnée

4.2.1 Schéma conceptuel

Schéma UserData :

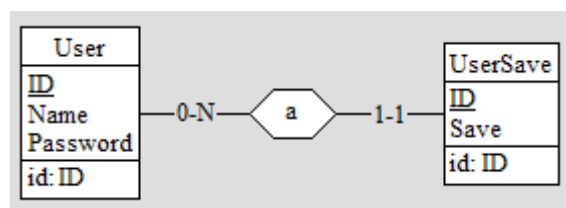
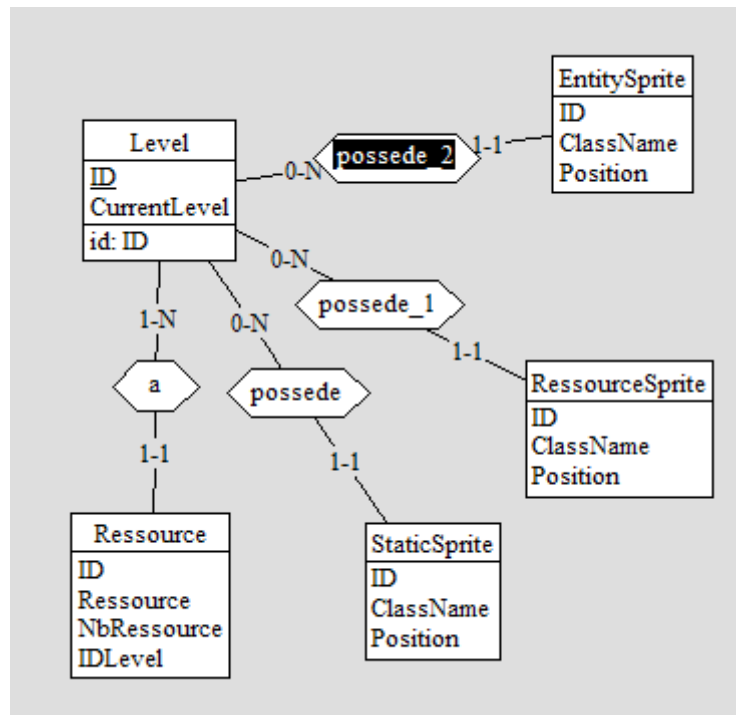


Schéma Save :



4.2.2 Schéma relationnel

Schéma UserData :

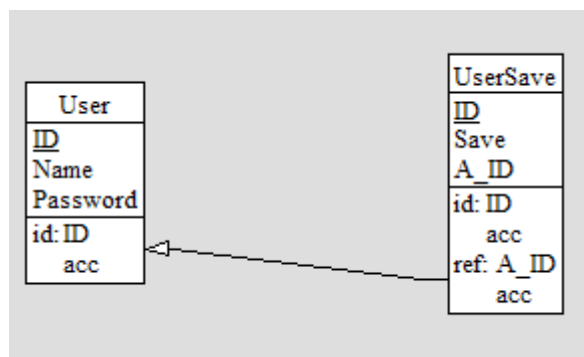
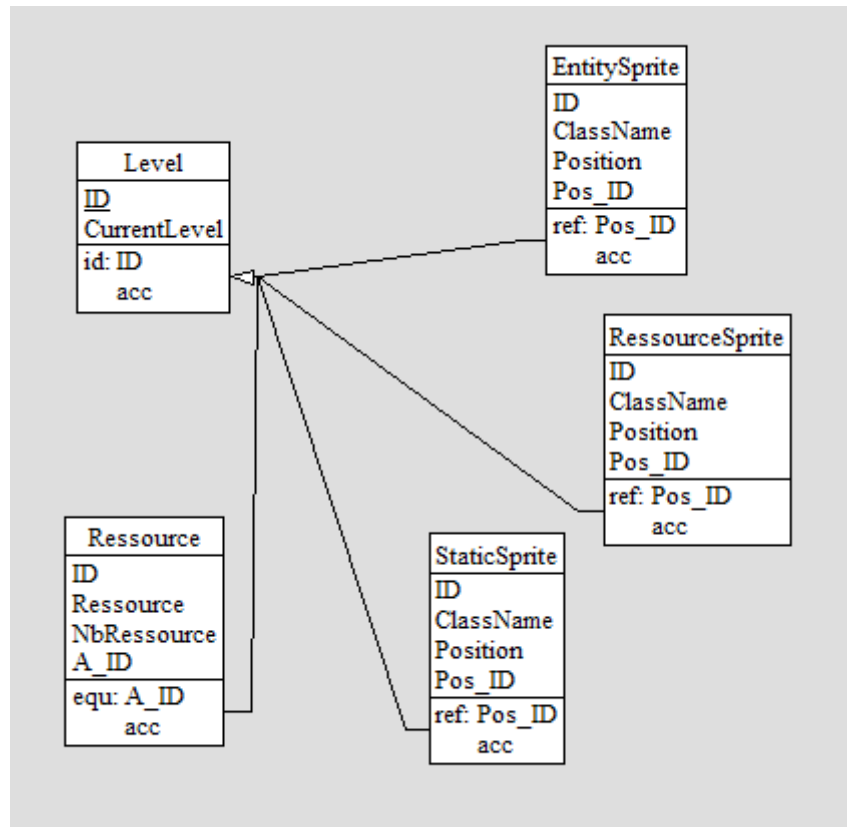


Schéma Save :



4.2.3 Contraintes d'intégrité

- Base de donnée des sauvegarde
 - Table Level :
 - ID : clé primaire
 - CurrentLevel : Chaîne de caractères
 - Table EntitySprite, RessourceSprite, StaticSprite :
 - ID : clé primaire
 - ClassName : Chaîne de caractères
 - Position : Chaîne de caractère
 - IdLevel : clé étrangère de la Table Level
- Base de donnée des utilisateur
 - Table User :
 - ID : clé primaire
 - Name : Chaîne de caractères

- Password : Chaîne de caractères
- Table UserSave :
 - ID : clé primaire
 - Save : chaîne de caractères
 - UserID : clé étrangère de la Table User

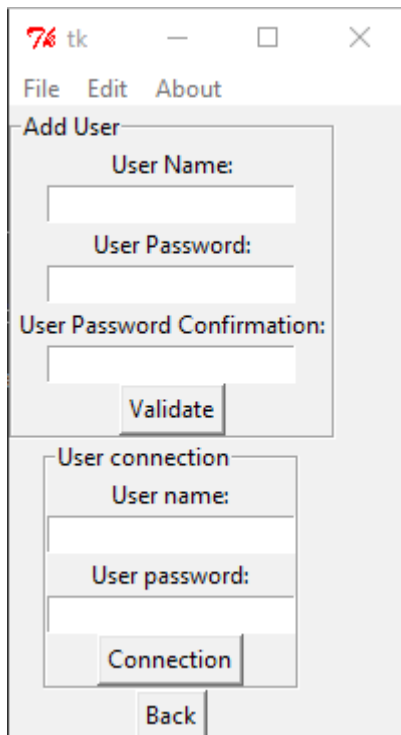
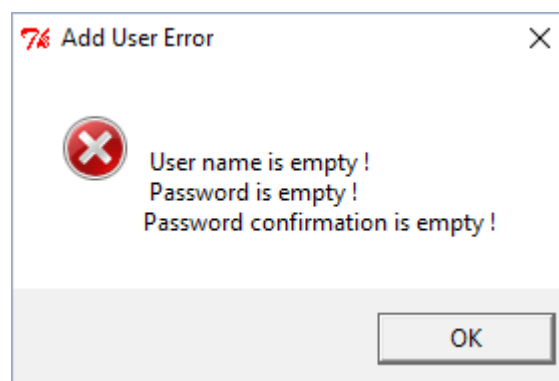
5 Mode d'emploi du programme

Depuis la fenêtre d'accueil nous avons accès à la création d'un utilisateur et de sa connexion. Une fois connecté, on a le choix de commencé une nouvelle partie ou d'en chargé une nouvelle. Arrivé en jeu, plusieurs action sont possible :

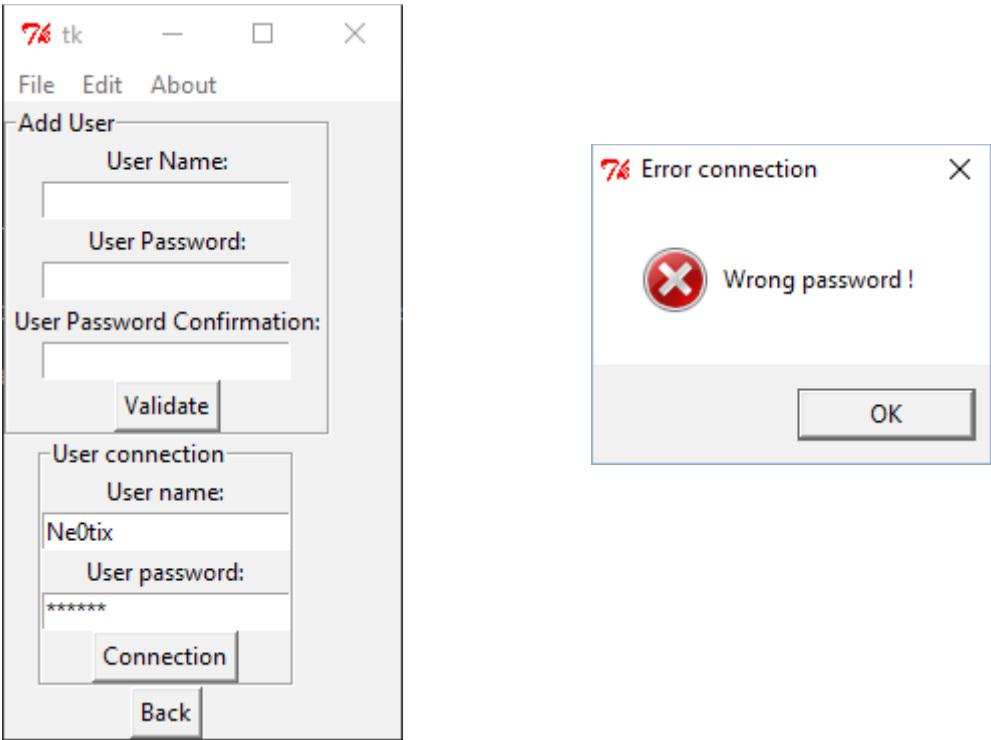
- Sélectionné une entité pour le déplacé
- Sélectionné une entité pour lui demander de récolter une ressource
- Sélectionné une entité pour lui faire construire un nouveau bâtiment.

6 Cas de tests

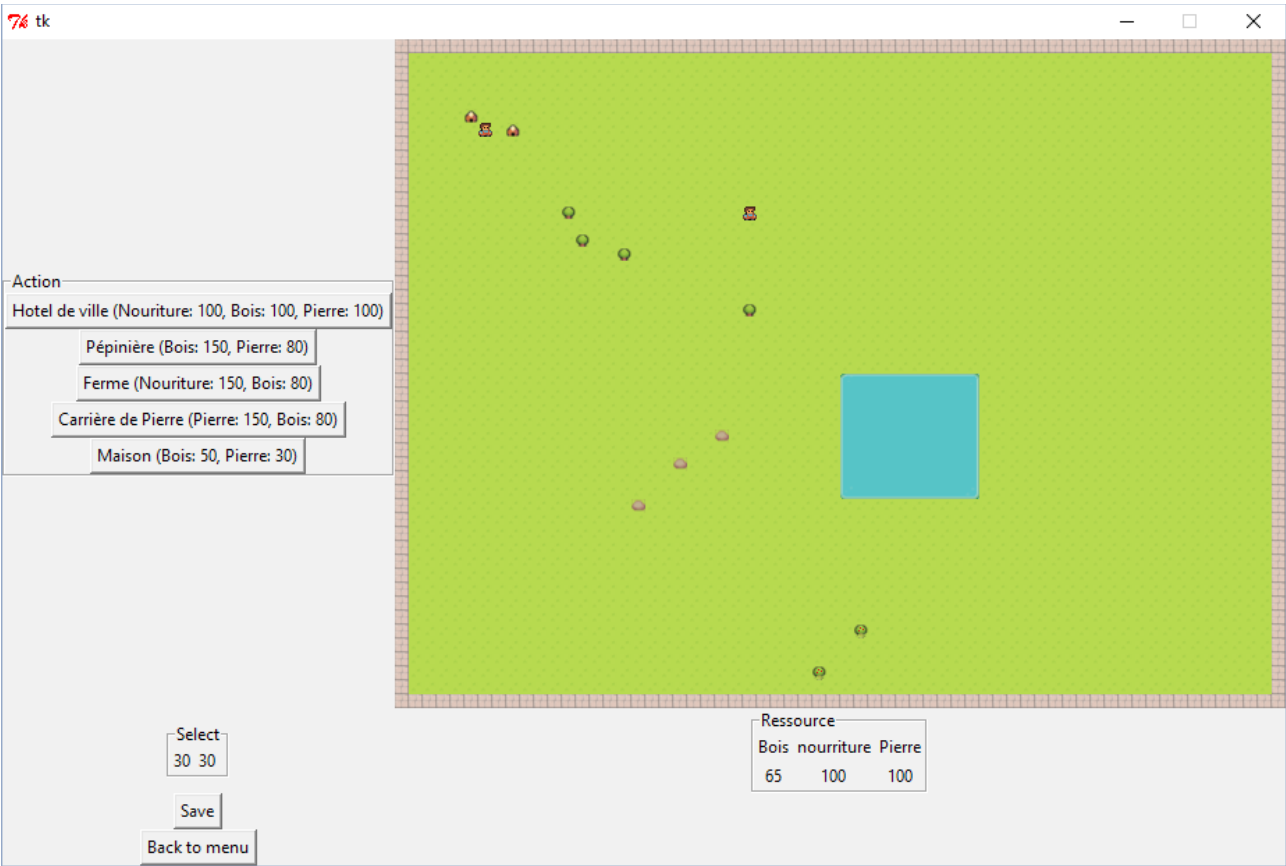
Erreur à la création d'un utilisateur

Erreur à la connexion



Déplacement d'une entité





7 Gestion des bugs

- Le bouton de chargement des sauvegardes génère une erreur lors qu'il n'y a pas de sauvegarde sélectionnée
- Le jeu retourne une erreur lorsque l'on veut déplacer une entité sur un emplacement impossible
- Le jeu se fige et s'arrete lors de la sauvegarde.
- Erreur, a la creation d'une utilisateur si il existe déjà

8 Avis des personnes de référence

9 Code Python

9.1 Modules MainMenu

```
from tkinter import *
from tkinter.messagebox import *
import sqlite3
import os.path
```

```

import TFE

dirBase = os.path.dirname(os.path.abspath(__file__))
db_path = os.path.join(dirBase, "DataBase/UserData.db3")

class dropDown(object):
    def __init__(self, screen):
        self.screen = screen
        menu = Menu(self.screen)

        # Sous Menu Fichier
        fileMenu = Menu(menu, tearoff=0)
        fileMenu.add_command(label="Main Menu", command= lambda: self.goToMainMenu())
        fileMenu.add_command(label="User Menu", command= lambda: self.goToUserMenu())
        fileMenu.add_separator()
        fileMenu.add_command(label="Exit", command= lambda: self.exit())

        menu.add_cascade(label="File", menu=fileMenu)

        # Sous Menu Edit
        editMenu = Menu(menu, tearoff=0)
        editMenu.add_command(label="Add User", command= lambda: self.goToAddUser())
        editMenu.add_command(label="Connection", command= lambda: self.goToConnection())

        menu.add_cascade(label="Edit", menu=editMenu)

        # Sous Menu About
        aboutMenu = Menu(menu, tearoff=0)
        aboutMenu.add_command(label="Ecole", command= lambda: self.goToEcole())
        aboutMenu.add_command(label="Createur", command= lambda: self.goToCreateur())

        menu.add_cascade(label="About", menu=aboutMenu)

        # Ajout a la fenetre
        screen.config(menu=menu)
    def childFrameDestroy(self):
        for child in self.screen.winfo_children():
            child.destroy()

    def goToMainMenu(self):
        self.childFrameDestroy()
        self.mainMenu = mainMenu(self.screen)

    def goToUserMenu(self):
        self.childFrameDestroy()
        self.userMenu = userMenu(self.screen)

    def goToGameMenu(self, userName):
        self.childFrameDestroy()
        self.gameMenu = gameMenu(self.screen, userName)

    def exit(self):
        self.screen.quit()
        self.screen.destroy()

    def goToAddUser(self):

```

```

self.childFrameDestroy()
self.addUser = addUser(self.screen)
self.backButton = Button(self.screen, text="Back", command= lambda: self.goToMainMenu())
self.backButton.grid()

def goToConnection(self):
    self.childFrameDestroy()
    self.userConnection = userConnection(self.screen)
    self.backButton = Button(self.screen, text="Back", command= lambda: self.goToMainMenu())
    self.backButton.grid()

def goToLoadSave(self, userName):
    self.childFrameDestroy()
    self.loadSave = loadSave(self.screen, userName)

def goToEcole(self):
    pass

def goToCreateur(self):
    pass

class mainMenu(dropDown):

    def __init__(self, screen):
        dropDown.__init__(self, screen)

        self.mainMenuFrame = Frame(screen, height="1024", width="640")
        self.mainMenuFrame.grid()
        # Titre
        self.labelMainTitle = Label(self.mainMenuFrame, text="Empires Of Wind")
        self.labelMainTitle.grid(row=0, column=0)

        # Bouton vers la fenetre user
        self.buttonSinglePlayer = Button(self.mainMenuFrame, text="Single Player", command= lambda: self.goToUserMenu())
        self.buttonSinglePlayer.grid(row=1, column=0)

        # Bouton pour sortir du jeu
        self.buttonExitGame = Button(self.mainMenuFrame, text="Exit Game", command= lambda: self.exit())
        self.buttonExitGame.grid(row=2, column=0)

class addUser(dropDown):
    def __init__(self, screen):
        dropDown.__init__(self, screen)

        self.addUserFrame = LabelFrame(screen, text="Add User")
        self.addUserFrame.grid()

        self.userNameLabel = Label(self.addUserFrame, text="User Name:")
        self.userNameLabel.grid()

        self.userNameEntry = Entry(self.addUserFrame)
        self.userNameEntry.grid()

        self.userPasswordLabel = Label(self.addUserFrame, text="User Password:")
        self.userPasswordLabel.grid()

```

```

self.userPasswordEntry = Entry(self.addUserFrame, show="*")
self.userPasswordEntry.grid()

self.userPasswordConfirmationLabel = Label(self.addUserFrame, text="User Password Confirmation:")
self.userPasswordConfirmationLabel.grid()

self.userPasswordConfirmationEntry = Entry(self.addUserFrame, show="*")
self.userPasswordConfirmationEntry.grid()

self.userValidationButton = Button(self.addUserFrame, text="Validate", command= lambda: self.userValidation())
self.userValidationButton.grid()

def userValidation(self):
    self.userName = self.userNameEntry.get()
    self.userPassword = self.userPasswordEntry.get()
    self.userPasswordConfirmation = self.userPasswordConfirmationEntry.get()
    self.userErrorList = ""

    if self.userName == "":
        self.userErrorList += "\n User name is empty !"
    if self.userPassword == "":
        self.userErrorList += "\n Password is empty !"
    if self.userPasswordConfirmation == "":
        self.userErrorList += "\nPassword confirmation is empty !"

    if self.userErrorList == "":
        if self.userPassword != self.userPasswordConfirmation:
            self.userErrorList += "\nPasswords doesn't match"
        if self.userName.find(" ") != -1:
            self.userErrorList += "\nNo space in user name"
        if self.userPassword.find(" ") != -1:
            self.userErrorList += "\nNo space in user password"

    if self.userErrorList != "":
        showerror("Add User Error", self.userErrorList)
    else:
        conn = sqlite3.connect(db_path)
        c = conn.cursor()
        c.execute("SELECT User from User where Name = ?", (self.userName,))
        self.user = c.fetchone()
        if self.user != None:
            showerror("Add User Error", "User name exist")
        else:
            c.execute("INSERT INTO User (Name, Password) VALUES (?,?);", (self.userName, self.userPassword))
            showinfo("Add User", "New user added")

        conn.commit()

        c.close()

class userConnection(dropDown):
    def __init__(self, screen):
        dropDown.__init__(self, screen)

    self.userConnectionFrame = LabelFrame(screen, text="User connection")

```

```

self.userConnectionFrame.grid()

self.connectionNameLabel = Label(self.userConnectionFrame, text="User name:")
self.connectionNameLabel.grid()

self.connectionNameEntry = Entry(self.userConnectionFrame)
self.connectionNameEntry.grid()

self.connectionPasswordLabel = Label(self.userConnectionFrame, text="User password:")
self.connectionPasswordLabel.grid()

self.connectionPasswordEntry = Entry(self.userConnectionFrame, show="*")
self.connectionPasswordEntry.grid()

self.connectionButton = Button(self.userConnectionFrame, text="Connection", command= lambda: self.connection())
self.connectionButton.grid()

def connection(self):
    self.connectionName = self.connectionNameEntry.get()
    self.connectionPassword = self.connectionPasswordEntry.get()

    self.connectionErrorList = ""

    if self.connectionName == "":
        self.connectionErrorList += "\n User name is empty !"
    if self.connectionPassword == "":
        self.connectionErrorList += "\n Password is empty !"

    if self.connectionErrorList == "":
        if self.connectionName.find(" ") != -1:
            self.connectionErrorList += "\nNo space in user name"
        if self.connectionPassword.find(" ") != -1:
            self.connectionErrorList += "\nNo space in user password"

    if self.connectionErrorList != "":
        showerror("Connection Error", self.connectionErrorList)
    else:
        conn = sqlite3.connect(db_path)
        c = conn.cursor()
        c.execute("SELECT Password from User where Name = ?", (self.connectionName,))
        self.userPasswordVerif = c.fetchone()
        conn.commit()

        c.close()
        if self.userPasswordVerif == None:
            showerror("Error connection", "Unknown user name !")
        else:
            if self.userPasswordVerif[0] != self.connectionPassword:
                showerror("Error connection", "Wrong password !")
            else:
                self.goToGameMenu(self.connectionName)

class userMenu(addUser, userConnection):
    def __init__(self, screen):
        addUser.__init__(self, screen)

```



```

userConnection.__init__(self, screen)
self.userMenuBackButton = Button(screen, text="Back", command= lambda: self.goToMainMenu())
self.userMenuBackButton.grid()

class gameMenu(dropDown):
    def __init__(self, screen, userName):
        dropDown.__init__(self, screen)
        self.userName = userName

        self.gameMenuFrame = LabelFrame(screen, text="Game")
        self.gameMenuFrame.grid()

        self.newGameButton = Button(self.gameMenuFrame, text="New Game", command= lambda: self.newGame())
        self.newGameButton.grid()

        self.loadGameButton = Button(self.gameMenuFrame, text="Load Game", command= lambda: self.goToLoadSave(self.userName))
        self.loadGameButton.grid()

        self.gameMenuBackButton = Button(self.gameMenuFrame, text="Back", command= lambda: self.goToUserMenu())
        self.gameMenuBackButton.grid()

    def newGame(self):
        self.screen.destroy()
        TFE.main(self.userName, False)

class loadSave(dropDown):
    def __init__(self, screen, userName):
        dropDown.__init__(self, screen)

        self.userName = userName

        self.loadSaveFrame = LabelFrame(screen, text="Load Save")
        self.loadSaveFrame.grid()

        self.saveListBox = Listbox(self.loadSaveFrame)

        conn = sqlite3.connect("DataBase/UserData.db3")
        c = conn.cursor()

        c.execute("SELECT Save from UserSave where UserID = (select ID from User where Name = (?)", ( self.userName,))
        for raw in c:
            self.saveListBox.insert(END, raw)
        conn.commit()
        c.close()

        self.saveListBox.grid()

        self.loadButton = Button(self.loadSaveFrame, text="Load", command= lambda: self.loadSaveGame(self.saveListBox.get(ACTIVE)))
        self.loadButton.grid()

        self.backButton = Button(self.loadSaveFrame, text="Back", command= lambda: self.goToGameMenu())
        self.backButton.grid()

    def loadSaveGame(self, save):
        self.screen.destroy()
        TFE.main(self.userName, True, save)

```

```

if __name__ == "__main__":
    master = Tk()
    menu = mainMenu(master)
    master.mainloop()

```

9.2 Cœur du jeu

```

from tkinter import *
from pygame.locals import *
import pygame
import constants
import os
import myIA
import random
import datetime
import sqlite3
import mainMenu

class main():

    def __init__(self, user, load=False, save=None):

        self.user = user
        self.saveName = save
        ##### Village Var #####

        self.game = Tk()
        self.game.resizable(width=False, height=False)

        # Frame Ressource
        self.ressource = LabelFrame(self.game, text="Ressource", width=500, height=150)
        self.ressource.grid(row=1, column=1)

        # Frame Select
        self.select = LabelFrame(self.game, text="Select", width=100, height=150)
        self.select.grid(row=1, column=0)

        # Frame Action
        self.action = LabelFrame(self.game, text="Action", width=100, height=480)
        self.action.grid(row=0, column=0)

        # Autre
        self.saveButton = Button(self.game, text="Save", command= lambda: self.mSave())
        self.saveButton.grid()

        self.backToMainmenu = Button(self.game, text="Back to menu", command= lambda: self.backToMainMenu())
        self.backToMainmenu.grid()

        # Frame Game
        self.embed = Frame(self.game, width=640, height=480)
        self.embed.grid(row=0, column=1)

        # Init de pygame dans la frame Game
        os.environ['SDL_WINDOWID'] = str(self.embed.winfo_id())

```

```

pygame.display.init()
self.screen = pygame.display.set_mode(constants.TAILLE_FENETRE)

if load:
    conn = sqlite3.connect("DataBase/"+self.saveName[0]+".db3")
    c = conn.cursor()

    c.execute("SELECT CurrentLevel from Level")
    for row in c:
        self.current_level = eval(row[0]+"(self.screen)")
        for objet in self.current_level.ressourceSprite:
            self.current_level.ressourceSprite.remove(objet)

        for objet in self.current_level.staticSprite:
            self.current_level.staticSprite.remove(objet)

        for objet in self.current_level.entitySprite:
            self.current_level.entitySprite.remove(objet)

    c.execute("SELECT ClassName, Position from RessourceSprite")
    for row in c:
        x = eval(row[0]+"(self.current_level, eval(row[1]))")
        self.current_level.ressourceSprite.add(x)

    c.execute("SELECT className, Position from StaticSprite")
    for row in c:
        x = eval(row[0]+"(self.current_level, eval(row[1]))")
        self.current_level.staticSprite.add(x)

    c.execute("SELECT className, position from entitySprite")
    for row in c:
        x = eval(row[0]+"(self.current_level, eval(row[1]))")
        self.current_level.entitySprite.add(x)

    conn.commit()
    c.close()
else:
    self.current_level = Level01(self.screen)

self.clock = pygame.time.Clock()
self.second = 0
pygame.time.set_timer(USEREVENT + 1, 300) # Timer déplacement
pygame.time.set_timer(USEREVENT + 2, 500) # Timer Action
pygame.time.set_timer(USEREVENT + 3, 1500)
self.actionToDo = {}
self.currentSelected = None

#### Main Loop ####
while True:

    for item in self.ressource.wininfo_children(): # Destruction de l'intérieur de la frame ressource
        item.destroy()

    ### Partie Gestion de la Frame Ressource ###
    keys = []

```

```

values = []

### Recuperation des informations et valeurs des ressource utilisé
for key in self.current_level.ressource.keys():
    keys.append(key)

for value in self.current_level.ressource.values():
    values.append(value)

### Affichage des ressource
for position in range(len(keys)):
    labelname = Label(self.ressource, text=keys[position])
    labelname.grid(row=0, column=position)

    labelitem = Label(self.ressource, text=values[position])
    labelitem.grid(row=1, column=position)

### Gestion des évènement ###
for event in pygame.event.get():

    if event.type == pygame.QUIT: # On quit game
        pygame.quit()
        self.game.quit()
        sys.quit()
    if event.type == pygame.MOUSEBUTTONDOWN:
        if pygame.mouse.get_pressed()[2]: # Clic droit
            if self.currentSelected != None:
                if str(self.currentSelected[1]) == "villageoi":
                    pos =pygame.mouse.get_pos()
                    self.chemin = self.IA(((self.currentSelected[0].positionX), (self.currentSelected[0].positionY)),
(int(((pos[0])/10)),int(((pos[1])/10))))
                    self.actionToDo[self.currentSelected[0]] = self.chemin
            if pygame.mouse.get_pressed()[0]: # Clic gauche

```

```

c.close()

def backToMainMenu(self):
    self.game.destroy()
    master = Tk()
    menu = mainMenu.mainMenu(master)
    master.mainloop()
class Level(object):
    def __init__(self):
        self.wallSpriteColide = pygame.sprite.Group()
        self.movingSprite = pygame.sprite.Group()
        self.staticSprite = pygame.sprite.Group()
        self.entitySprite = pygame.sprite.Group()
        self.ressourceSprite = pygame.sprite.Group()
        self.ressource = {}

    def update(self):
        self.wallSpriteColide.update()
        self.movingSprite.update()
        self.staticSprite.update()
        self.entitySprite.update()
        self.ressourceSprite.update()

    def drawing(self, screen):
        screen.fill((0, 255, 0))
        screen.blit(self.background, (0,0))

        self.wallSpriteColide.draw(screen)
        self.staticSprite.draw(screen)
        self.ressourceSprite.draw(screen)
        self.entitySprite.draw(screen)
        self.movingSprite.draw(screen)

class Level01(Level):
    def __init__(self, screen):
        Level.__init__(self)
        self.screen = screen

        self.background = pygame.image.load(constants.imgBackground).convert()
        self.background.set_colorkey((0, 255, 0))

        levelInit = [
            [constants.imgWall640, 0, 0],
            [constants.imgWall640, 0, 47],
            [constants.imgWall480, 0, 0],
            [constants.imgWall480, 63, 0],
            [constants.imgLac, 32, 24],
        ]

        for objet in levelInit:
            block = sprite(objet[0])
            block.rect.x = objet[1]*10
            block.rect.y = objet[2]*10
            self.wallSpriteColide.add(block)

        levelRessource = [

```

```

        [bois, (16, 15)],
        [bois, (25, 19)],
        [pierre, (20, 30)],
        [pierre, (23, 28)],
        [pierre, (17, 33)],
        [nourriture, (30, 45)],
        [nourriture, (33, 42)]
    ]

    for ressource in levelRessource:
        x = ressource[0](self, ressource[1])
        self.ressourceSprite.add(x)

    self.home1 = home(self, (5,5))
    self.staticSprite.add(self.home1)
    self.villageoi1 = villageoi(self, (6,6))
    self.entitySprite.add(self.villageoi1)

    self.home2 = home(self, (8, 6))
    self.staticSprite.add(self.home2)
    self.villageoi2 = villageoi(self, (9, 7))
    self.entitySprite.add(self.villageoi2)

class entity(pygame.sprite.Sprite):
    def __init__(self, level, position):
        pygame.sprite.Sprite.__init__(self)
        self.currentLevel = level
        self.positionX = position[0]
        self.positionY = position[1]

    def loadImg(self, img, position):
        self.image = pygame.image.load(img).convert_alpha()
        self.rect = self.image.get_rect()
        self.rect.x = position[0]*10
        self.rect.y = position[1]*10

    def move(self, pos):
        self.positionX = (pos[0])
        self.positionY = (pos[1])
        self.rect.x = self.positionX*10
        self.rect.y = self.positionY*10

### Class de tous les batiment et leur fonctionnement
class batiment(entity):
    def __init__(self, level, position):
        entity.__init__(self, level, position)
        self.cost = {}
        self.needProduct = {}
        self.makeProduct = {}
        self.travailleur = {}
        self.coef = 1

class home(batiment):
    def __init__(self, level, position):
        batiment.__init__(self, level, position)
        self.cost = {"Bois": 50, "Pierre": 30}

```

```

        self.needProduct = {"Bois": 1}
        self.loadImg(constants.imgHome, position)

def drawAction(self, fen):
    pass

class comptoire(batiment):
    def __init__(self, level, position):
        batiment.__init__(self, level, position)
        self.cost = {"nourriture" : 100, "Bois": 100, "Pierre":100}
        self.needProduct = {"Pierre" : 5}
        self.loadImg(constants.imgComptoire, position)

    def drawAction(self, fen):
        Label.test = Label(fen.action, text="Hotel de ville")
        Label.test.grid(row=0)

class woodFarm(batiment):
    def __init__(self, level, position):
        batiment.__init__(self, level, position)
        self.cost = {"Bois" : 150, "Pierre": 80}
        self.makeProduct = {"Bois" : 3}
        self.loadImg(constants.imgWoodFarm, position)

class wheatFarm(batiment):
    def __init__(self, level, position):
        batiment.__init__(self, level, position)
        self.cost = {"nourriture": 150, "Bois" : 80}
        self.makeProduct = {"nourriture" : 3}
        self.loadImg(constants.imgWheatFarm, position)

class carrier(batiment):
    def __init__(self, level, position):
        batiment.__init__(self, level, position)
        self.cost = {"Pierre": 150, "Bois": 80}
        self.makeProduct = {"pierre" : 3}
        self.loadImg(constants.imgCobleFarm, position)

### class de tous les ressources et leur fonctionnement
class ressource(entity):
    def __init__(self, level, position):
        entity.__init__(self, level, position)
        self.currentRessource = 300
        self.ressource = 300

class pierre(ressource):
    def __init__(self, level, position):
        ressource.__init__(self, level, position)
        self.loadImg(constants.imgPierre, position)
        self.product = "Pierre"
        level.ressource[self.product] = 100

    def drawAction(self, fen):
        self.nameRessouceLabel = Label(fen.select, text="Pierre")
        self.nameRessouceLabel.grid(row=0, column=0)

```

```

self.ressourceLabel = Label(fen.select, text="Ressource :")
self.ressourceLabel.grid(row=1, column=0)

self.currentRessourceLabel = Label(fen.select, text=self.currentRessource)
self.currentRessourceLabel.grid(row=1, column=1)

```

```

class bois(ressource):
    def __init__(self, level, position):
        ressource.__init__(self, level, position)
        self.loadImg(constants.imgArbre, position)
        self.product = "Bois"
        level.ressource[self.product] = 100

    def drawAction(self, fen):
        self.nameRessouceLabel = Label(fen.select, text="Bois")
        self.nameRessouceLabel.grid(row=0, column=0)

        self.ressourceLabel = Label(fen.select, text="Ressource :")
        self.ressourceLabel.grid(row=1, column=0)

        self.currentRessourceLabel = Label(fen.select, text=self.currentRessource)
        self.currentRessourceLabel.grid(row=1, column=1)

```

```

class nourriture(ressource):
    def __init__(self, level, position):
        ressource.__init__(self, level, position)
        self.loadImg(constants.imgnourriture, position)
        self.product = "nourriture"
        level.ressource[self.product] = 100

    def drawAction(self, fen):
        self.nameRessouceLabel = Label(fen.select, text="nourriture")
        self.nameRessouceLabel.grid(row=0, column=0)

        self.ressourceLabel = Label(fen.select, text="Ressource :")
        self.ressourceLabel.grid(row=1, column=0)

        self.currentRessourceLabel = Label(fen.select, text=self.currentRessource)
        self.currentRessourceLabel.grid(row=1, column=1)

```

```

class sprite(pygame.sprite.Sprite):

    def __init__(self, imageData=None):
        pygame.sprite.Sprite.__init__(self)

        if imageData != None:
            self.image = pygame.image.load(imageData).convert_alpha()
            self.rect = self.image.get_rect()

        else:

```



```

        self.image = pygame.Surface((10,10))
        self.rect = self.image.get_rect()

def moveSprite(self, position):
    self.rect.x = position[0]
    self.rect.y = position[1]

class villageoi(entity):
    def __init__(self, level, position):
        entity.__init__(self, level, position)
        self.vie = 30
        self.vieMax = 30
        self.loadImg(constants.imgChar, position)
        self.moveActive = False

def drawAction(self, fen):
    # Frame Action
    comptoireButton = Button(fen.action, text="Hotel de ville (Nouriture: 100, Bois: 100, Pierre: 100)", command= lambda: self.buildComptoire())
    comptoireButton.grid(row=0)

    woodFarmButton = Button(fen.action, text="Pépinère (Bois: 150, Pierre: 80)", command= lambda: self.buildWoodFarm())
    woodFarmButton.grid(row=1)

    wheatFarmButton = Button(fen.action, text="Ferme (Nouriture: 150, Bois: 80)", command= lambda: self.buildWheatFarm())
    wheatFarmButton.grid(row=2)

    cobleFarmButton = Button(fen.action, text="Carrière de Pierre (Pierre: 150, Bois: 80)", command= lambda: self.buildCobleFarm())
    cobleFarmButton.grid(row=3)

    homeButton = Button(fen.action, text="Maison (Bois: 50, Pierre: 30)", command= lambda: self.buildHome())
    homeButton.grid(row=4)

    # Frame Select
    VieLabel = Label(fen.select, text=self.vie)
    VieMaxLabel = Label(fen.select, text=self.vieMax)
    VieLabel.grid(row=0, column=0)
    VieMaxLabel.grid(row=0, column=1)

def moveEntity(self):
    self.moveActive = True

def buildComptoire(self):
    x =comptoire(self.currentLevel, (10,10))
    validator = True
    for price in x.cost:
        if self.currentLevel.ressource[price] - x.cost[price] < 0:
            validator = False
    if validator:
        self.currentLevel.movingSprite.add(x)

def buildHome(self):
    x = home(self.currentLevel, (10,10))
    validator = True
    for price in x.cost:
        if self.currentLevel.ressource[price] - x.cost[price] < 0:
            validator = False

```

```

    if validator:
        self.currentLevel.movingSprite.add(x)
def buildWoodFarm(self):
    x = woodFarm(self.currentLevel, (10,10))
    validator = True
    for price in x.cost:
        if self.currentLevel.ressource[price] - x.cost[price] < 0:
            validator = False
    if validator:
        self.currentLevel.movingSprite.add(x)

def buildWheatFarm(self):
    x = wheatFarm(self.currentLevel, (10,10))
    validator = True
    for price in x.cost:
        if self.currentLevel.ressource[price] - x.cost[price] < 0:
            validator = False
    if validator:
        self.currentLevel.movingSprite.add(x)

def buildCobleFarm(self):
    x = carrier(self.currentLevel, (10,10))
    validator = True
    for price in x.cost:
        if self.currentLevel.ressource[price] - x.cost[price] < 0:
            validator = False
    if validator:
        self.currentLevel.movingSprite.add(x)

```

10 Requêtes SQL

```

CREATE table User(
ID INTEGER NOT NULL unique primary key asc autoincrement,
Name TEXT unique,
Password TEXT);

```

```

create table UserSave(
ID INTEGER not null unique primary key asc autoincrement,
Save TEXT,
UserID REFERENCES User);

```

```

create table Level(
    ID integer not null unique primary key asc autoincrement,
    CurrentLevel TEXT
);

```

```

create table StaticSprite(
    ID integer not null unique primary key asc autoincrement,
    ClassName TEXT,
    Position TEXT,
    IDLevel REFERENCES Level
);

```

```

create table EntitySprite(
    ID integer not null unique primary key asc autoincrement,
    ClassName TEXT,

```

```
Position TEXT,  
IDLevel REFERENCES Level  
);  
  
create table RessourceSprite(  
ID integer not null unique primary key asc autoincrement,  
ClassName TEXT,  
Position TEXT,  
IDLevel REFERENCES Level  
);
```

11 Conclusion

Je suis arrivé a mon objectif dans se travaille, je savais dés le départ que je n’aurais pas le temps d’implémenté toutes les fonctionnalités que j’aurais voulus y mettre. La création d’un jeu de toutes pièces sans guides demande du temps, de l’expérience et pas mal de connaissance du langage. Trois éléments que je n’avais pas, j’ai donc passé beaucoup de temps a exploré. Je connaissais mon objectif finale mais je ne savais pas quels éléments de gameplay je devais utiliser ni comment. Ce projet m’a appris beaucoup de chose sur python et la création de jeux vidéo.