Projet en programmation

Groupe 6 – Terminus

Quentin DOLLEN
Antoine GRAND
Siméon GRAVIS
Pierre-Gilles LE BOTLAN
Phuoc LE DOAN
Chelsy MENTOU

Screenshot du jeu Caesar III Terminus (semaine 4 2023)

Avancées du projet :

- Interface graphique
    - Hud (Boutons d'ajout de bâtiments, Overlay)
    - Launcher du jeu
    - Sélection de la partie
- Logique
    - Logique des walker (déplacement)
    - Logique des bâtiment (walker associé)
- Affichage
    - Carte
    - Mini map
- Menus de sélections
    - Sélection sauvegardes
    - Sélection du nom du jeux
    - Menu d'accueil

Le modèle MVC :

Pour gérer notre projet, nous avons choisi de suivre le modèle MVC : Model-View-Controller. Nous sommes séparés en 3 groupes de 2 et notre mission fut d'implémenter pour chaque groupe les features attendues :

☐ **Partie Model**

Pierre-Gilles, Antoine et Quentin ont travaillé sur la partie Model. Nous avons développé la partie logique, à savoir les différentes classes du jeu, leurs méthodes, et les fonctions qui les font interagir mutuellement.

Nous avons choisi de modéliser le jeu sous forme de matrices: une matrice contient les bâtiments et une matrice contient les personnages. Toutes les fonctions implémentées devaient servir à soutenir trois fonctions principales: une qui déplace les walker, une qui réalise toutes les opérations cohérentes au jeu sur les bâtiments (par exemple la gestion du feu, de l'emploi, évolution des maisons…) et la troisième gère les opérations en lien avec les personnages (livraisons de marchandises, interaction avec les bâtiments…).

Un des problèmes rencontrés est dû à un manque de mise en commun ; certaines fonctions n'étaient pas sécurisées (peu), et des edge cases étaient atteints dans la mise en commun, là où dans les tests réalisés par ceux qui implémentaient ces méthodes ne couvraient pas toutes les possibilités, et cela a donc mené à des corrections de dernière minutes.

Nous avons également rencontré un problème de connaissances: au début du projet nous voulions modéliser des classes, gérer l'héritage, sans avoir vu en Java ces notions, donc le temps de les maîtriser, nous avons perdu un peu de temps.

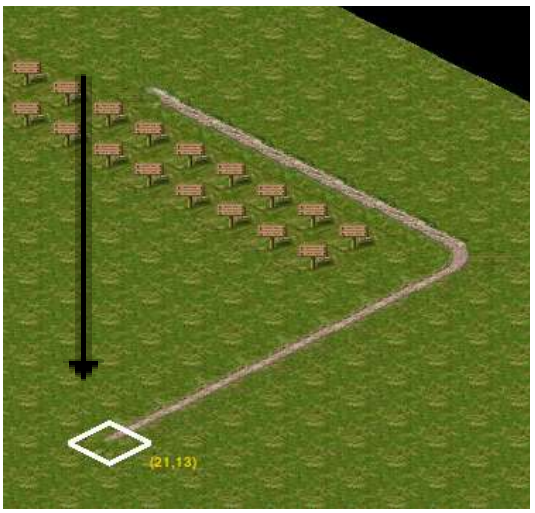
Nous avons correctement modélisé l'évolution des panneaux en maison, et les trois premières évolutions des maison, le feu, l'effondrement et la production de blé dans les fermes, qui est ensuite acheminé dans un entrepôt, le recrutement des citoyens par des recruteurs, l'arrivée des migrants, la collecte de nourriture par les marchés, et la redistribution ensuite dans les habitations.

En terme de répartition du travail, Pierre-Gilles s'est chargé du déplacement des personnages dans le cas où leur itinéraire est fixé, du pathfinding, d'une partie de la gestion des matrices, d'une partie de l'implémentation des classes, la plupart des tests réalisés en logique (sur les bâtiments et les personnages), Antoine s'est chargé d'une partie de l'implémentation des classes, d'une partie des tests logiques, d'une partie de la gestion des matrices. Pierre-Gilles et Antoine ont fait ensemble la destruction des bâtiments, gestion du feu, gestion de la destruction et de la suppression des personnages, le recrutement des citoyens, l'arrivée des migrants et l'évolution des maisons. Quentin a géré le déplacement aléatoire lors de croisement de routes. Le fonctionnement de Deplacement_basique() est plutôt simple. Cette fonction retourne la position suivante d'un walker passé en paramètre. Si ce walker n'as pas de destination précise (i.e. il n'a pas encore un pathfinding établi). Alors il va regarder ses champs des possibles. On stocke dans la classe walker sa position précédente pour éviter que le walker face des allez retour en boucle. Lors d'un croisement il prend un chemin parmi ceux qu'il n'a pas encore empruntés.

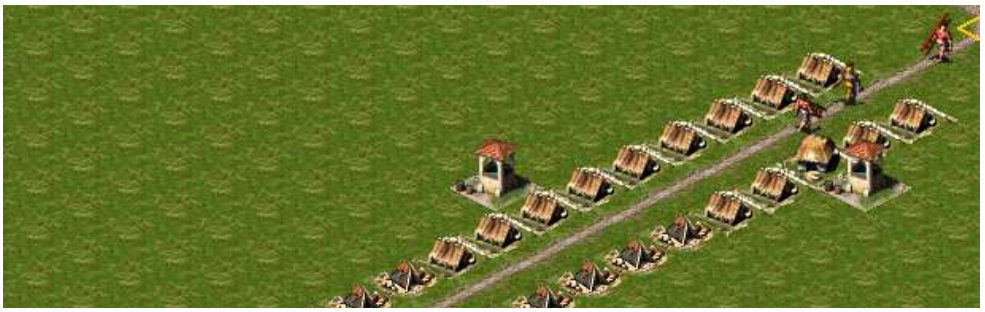
Quentin a géré les implémentations liées aux réactions logiques concernant l'appui des boutons: construire un puits, une caserne, un poste d'ingénieur, une ferme, Pierre Gilles a géré la destruction de bâtiments sur une certaine zone, la construction de maisons sur une zone, et la construction des routes en coudes. Il a fait la gestion de la matrice contenant toutes les cases alimentés en eau.

Siméon a très partiellement contribué à la logique, pour créer des fonctions lui permettant de récupérer les informations dont il avait besoin à l'affichage du jeu.

Quelques exemples de ce que la logique permet de faire :



Exemple de création de route en coude (flèche = déplacement de la souris)


Exemple d'évolution de maison avec accès à l'eau

Trois “tests” successifs sont réalisés :
– un test portant sur les personnages
– un test portant sur les bâtiments
– une fonction déplaçant tous les personnages.

De cette façon, dans la boucle de jeu, tous les personnages sont déplacés, puis tous les bâtiments sont testés, la fonction de test permettant leurs éventuelles actions requises d’être effectuées. La même chose est ensuite faite avec les personnages, puis la boucle recommence.

Dans ces tests logiques, on vérifie les conditions d’appel d’autres méthodes, de façon cohérente au jeu. Par exemple, dans le cas de la dame du marché, si une maison se trouve à proximité d’elle, elle procède à une distribution de nourriture. La fonction distribution n’aurait pas de sens à être appelée ailleurs.

Une des fonctions les plus utilisées dans ces tests est la fonction get_bat_prox(), écrite par Pierre-Gilles, qui renvoie dans un tableau l’ensemble des bâtiments intéressants situé autour d’un point dans un certain rayon. Cela nous permet par exemple, de savoir si un personnage est à proximité de son bâtiment cible quand il en a un, au préfet de baisser le risque de feu des bâtiments autour de lui.

On a encore des erreurs au niveau de l'exécution: Concernant la suppression des bâtiments, le code logique fonctionne, mais la fonction est parfois appelée sur des bâtiments non ciblés, en raison de problèmes non identifiés.

☐ **Partie View**

Siméon et Phuoc se sont chargés de la partie View. Notre objectif était de développer la partie graphique du jeu (hors Interface). Cela comprend l'affichage de la carte, de la mini map, des tuiles et aussi gérer la caméra lors des déplacements de souris.

Nous sommes partis de tutoriels sur internet. Il y a des similitudes dans l'architecture du code d'affichage avec celui de ces tutoriels. Nos compétences en pygame et en Infographie (voire en OOP en Septembre) n'étaient pas suffisamment approfondies pour pouvoir créer un code 100% fonctionnel dès octobre.

Phuoc a réalisé la carte afin que nous puissions placer des objets (arbres, rivières, bâtiments) de manière fixe, par l'idée d'utiliser plusieurs matrices 2D pour les afficher. Nous avons une matrice pour le terrain, et une matrice qui ne sert qu'aux textures de type nature (arbre, roche, eau), afin de retrouver la bonne texture pour le bon arbre à chaque frame. Il y a aussi une autre matrice d'affichage qui fait le lien avec la logique. La séparation des tâches au début du projet à fait que la matrice Affichage et la matrice des bâtiments de la partie Model sont différentes.

create_map() est la fonction qui renvoie toutes les informations importantes pour le placement des tuiles sur le damier isométrique (Position de rendu, image à placer …). Elle est appelée avant la fonction draw() (qui affiche les carreaux à l'écran). A cause de la séparation des tâches et du manque d'un main jusqu'à très tard dans le projet, la fonction create_map() (et sa soeur create_walkeur()) non pas plus être testé à temps, et nous nous sommes aperçu du problème majeur de l'affichage bien trop tard :

L'optimisation. Les 2 fonctions étant appelées à chaque tick de l'horloge, elles ralentissent beaucoup le jeu. Optimiser ce code aurait demandé de repenser l'architecture du jeu et la manière d'afficher, ce qui n'était pas envisageable vers la fin du projet.

La fonction grid_to_map(): Tous les éléments nécessaires pour créer la carte et la minimap sont dans cette fonction. Elle accueille les indices de la matrice vers les bonnes textures (il y a des dizaines de textures différentes pour les tuiles de nature, d'où la nécessité d'une seconde matrice pour s'y retrouver (matrixNature).

Nous avons (Phuoc) implémenté une caméra pour la carte. Nous utilisons la fonction pygame.mouse.get_pos() pour obtenir la position du curseur de la souris et nous mettons les conditions et utilisons une variable "self.speed" (dans le code : self.speed = 23) pour contrôler la vitesse de la souris.

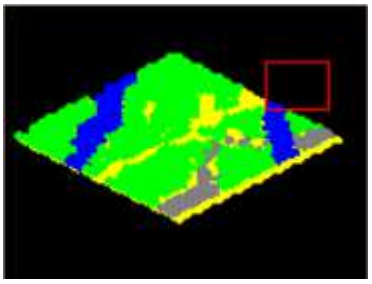
```
mouse_pos = pg.mouse.get_pos()
# x movement
if mouse_pos[0] > self.width * 0.98:
    self.dx = -self.speed
elif mouse_pos[0] < self.width * 0.02:
    self.dx = self.speed
else:
    self.dx = 0

# y movement
if mouse_pos[1] > self.height * 0.98:
    self.dy = -self.speed
elif mouse_pos[1] < self.height * 0.02:
    self.dy = self.speed
else:
    self.dy = 0

# update camera scroll

# camera-x
self.scroll.x += self.dx

# camera-y
self.scroll.y += self.dy
```

Ensuite, nous avons implémenté la limitation des bords de la zone de jeu pour la caméra, afin d'éviter de partir indéfiniment dans une des directions

```
# limitation for scrolling camera-x
if self.scroll.x < -1220:
    self.scroll.x = -1220
if self.scroll.x > 600:
    self.scroll.x = 600
```

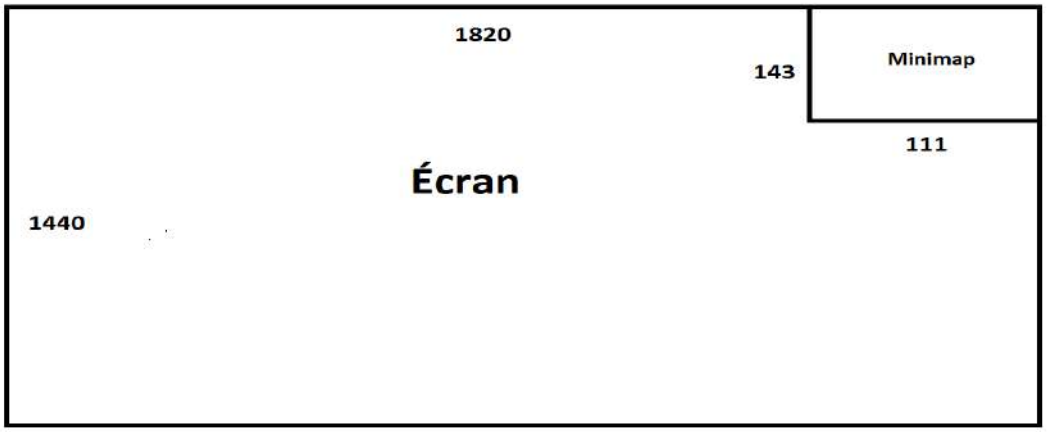
```
# limitation for scrolling camera-y
if self.scroll.y < -850:
    self.scroll.y = -850
if self.scroll.y > 550:
    self.scroll.y = 550
```

La mini-carte (qui a aussi été réalisée par Phuoc) marche de manière similaire à la carte. La mini-carte est créée avec des points de couleur représentant les différents objets sur la carte, similairement à Caesar III. Nous avons la mini-carte ci-dessous :



Pour une meilleure utilisation de la minimap, on a créé un pointeur sur la minicarte afin que nous puissions facilement reconnaître notre position à l'écran dans le jeu, avec le point représentant la position de la souris sur l'écran, et le cadre rectangulaire autour est la zone autour du curseur circulaire, comme ci-dessous (le point n'est pas dans la version finale):

Pour contrôler la vitesse du cadre sur le minimap, on va utiliser un factor x et un factor y entre l'écran et la mini-carte, en particulier self.factorx = (143*1.0) / 1820, self.factory = (111*1.0) / 1400



Ensuite, on va calculer la vitesse du mini-map comme ci-dessous:

```
# update camera scroll

# camera-x
self.scroll.x += self.dx
self.scroll_mini.x += self.dx * self.factorx
# camera-y
self.scroll.y += self.dy
self.scroll_mini.y -= self.dy * self.factory
```

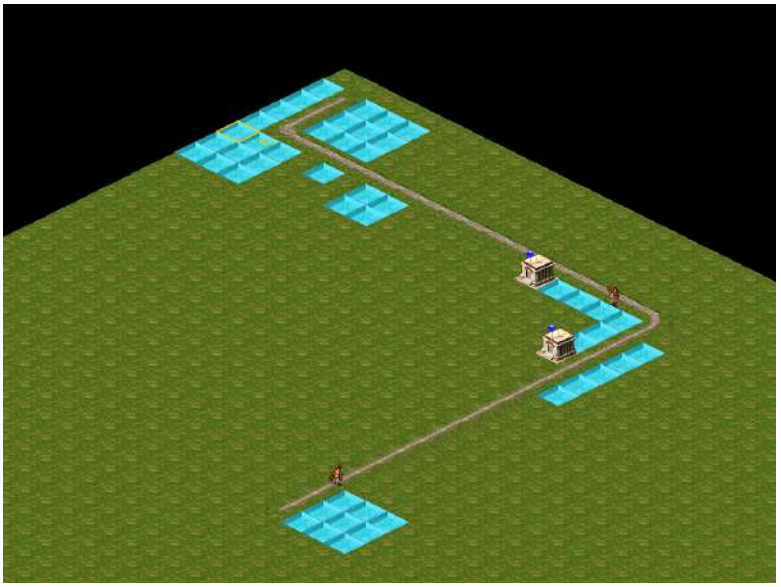
Et on a mis la limitation pour le mini-camera (similaire avec écran) comme ci-dessous:

```
# limitation for scrolling camera-mini-x
if self.scroll_mini.x < -144.3 + 26:
    self.scroll_mini.x = -144.3 + 26
if self.scroll_mini.x > 0:
    self.scroll_mini.x = 0
# limitation for scrolling camera-mini-y
if self.scroll_mini.y > 111 - 20:
    self.scroll_mini.y = 111 - 20
if self.scroll_mini.y < 0:
    self.scroll_mini.y = 0
```

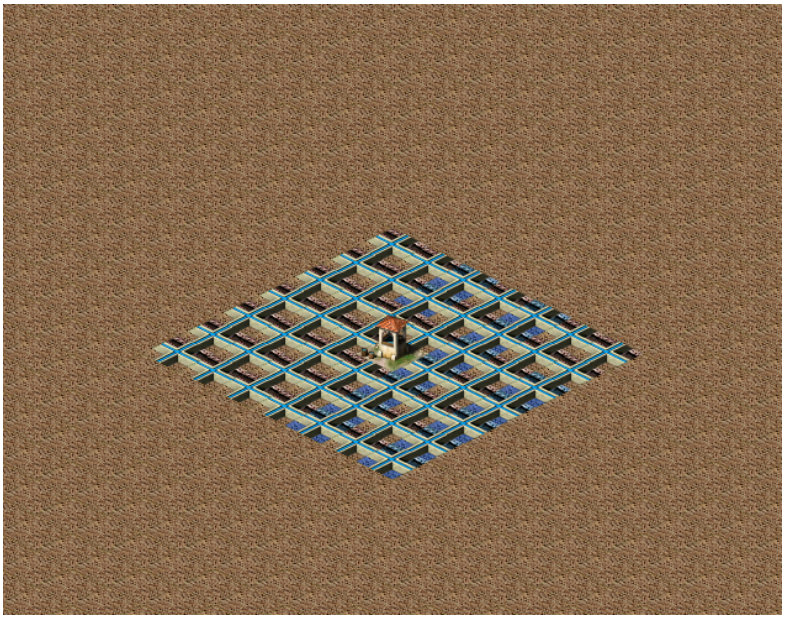
Les walkers marchent presque de la même manière que nos tuiles. Sur la case X,Y du damier, le walker est affiché après la tuile, pour être sûr que la texture de la tuile soit derrière le walker et que la tuile de la case X+1,Y+1 recouvre bien le walker si elle est grande (arbre, grand bâtiment). Les animations n'ont pas été implémentées à cause de la manière matricielle dont sont géré les walkers en partie Model.

Une version non fonctionnelle de l'overlay est visible dans les fonctions grid_to_map() et grid_to_walkeur() (Nous vous conseillons de ne pas jeter un coup d'oeil à ces fonctions, elles ne sont VRAIMENT pas agréable à l'oeil). L'overlay fonctionne pour le feu, l'eau et l'effondrement.
En utilisant les blocs de couleurs de Caesar III (ceux utilisé pour rendre
compte de la désirabilité d'un lieu), nous montrons les indices d'effondrements des bâtiments, et aussi tous les bâtiments d'ingénierie.
Chaque bâtiment sera remplacé par un bloc de couleur. C'est simple, mais c'est parce que l'affichage de la map n'est pas optimisé. On se permet de renouveler la map à chaque tick.

Overlay d'effondrement



Overlay d’eau (rappelant Simcity 4)



Toute texture ne venant pas officiellement de la librairie de texture de Caesar III à été modifiée par Siméon. (Interface compris)

Nous n'avons pas fait les animations (walkers + bâtiments) donc les mouvements de walkers sont plus proches des voitures de Simcity 2000 que des mouvements fluides de Caesar III.

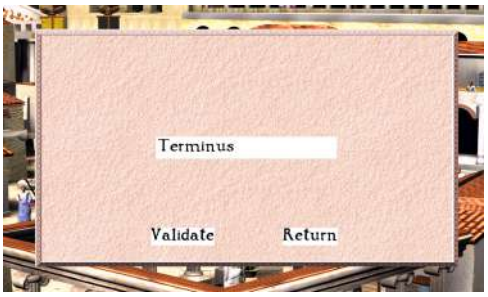
☐ **Partie Controller**

Dans cette partie, c’est Quentin et Chelsy qui y ont contribué. Cette partie est dédiée à toutes les intéractions entre le joueur et le jeu. La partie controller est divisée en plusieurs parties.

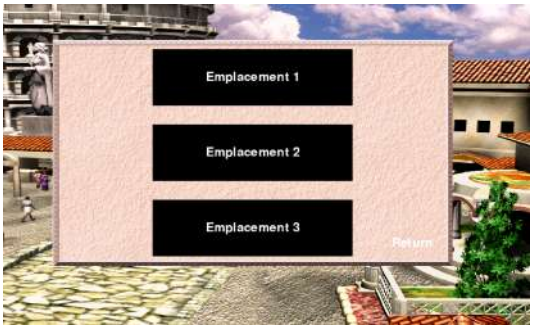
- [En général] Chaque page est une classe contenant une liste de boutons cliquables ou non. Quand les boutons sont survolé par la souris, ils sont mis en surbrillance. Quand un click de souris sur un bouton est ressenti par pygame. Alors une action est communiqué

- [Chelsy] La **page d'accueil**, mon premier travail dans le projet. J’ai fait la classe bouton qui permet détecter si la position de la souris est sur le bouton et de l'éclaircir.

- [Quentin] Le menu de **choix de nom de partie**. La réel difficultée a été de créer la classe Inputbox pour modifier le nom de la partie comme l'on souhaite



- [Quentin] Le menu de **sélection des parties**. Très facile a faire, le seul "Challenge" aura été d'utiliser pickle pour la sauvegarde et la recharge des parties et d'avoir un affichage approprié pour la bonne compréhension du joueur.



- [Quentin] Le **hud** lors des parties. Le plus dur aura été de placer les boutons dans les bons emplacements. Les boutons étant adaptables à la taille de l'écran du joueur, devoir voir tous les positionner en fonction de la taille de l'hud a été une prise de temps.

- [Quentin] Détection de la position de la souris. Lorsque l'on veut créer un pâté de maison, il faut trouver la position de la souris puis la convertir en un emplacement de case sur notre grille.
- Quentin a géré la **navigation** entre ces pages

Problèmes rencontrés au cours du projet

- **Utilisation de Github**

Durant le projet, la majorité du groupe a découvert l'outil merveilleux qu'est Git. Malheureusement, la gestion calamiteuse des branches, des répertoires locaux et des différents commits par les débutants du groupe (Siméon, Phuoc …) ont mené à bien des débâcles. Des après-midi entiers ont été consacrés à tenter de réparer ces erreurs de parcours.

- **Séparation des groupes**

Le projet à donc été séparé en 3 groupes distincts selon le modèle MVC (Model,View,Controller). Chacun des groupes s'est mis dans son coin, avec ses propres objectifs et sa propre vision du projet. Le manque de communication entre les groupes à mené à des problèmes de compatibilités entre les codes de chacun, et donc encore de longue session de débogage qui auraient pû être évitées.

Par exemple de ce manque de communication : la matrice de l'affichage n'est pas la même que celle de la logique, ce qui entraîne des pertes de performances et une chute de fps.

- **Expérience en Infographie**

Le groupe n'a pas d'expérience en infographie. Nous nous sommes entraînés avec quelques tutos pygame en septembre, mais ils ne furent pas suffisants pour maîtriser les aspects les plus compliqués, notamment les animations de walkers rendues quasi impossible à cause d'un manque de communication qui a mené à une implémentation logique ne permettant pas d'anticiper le déplacement et de représenter la trajectoire de façon propre.

Nous avons donc dû bricoler avec ce que nous savions faire à l'époque (Septembre) pour commencer l'implémentation en hâte.

Leçons à tirer de ce projet (Conclusion):

-Utilisation de git : Le faire correctement et rigoureusement, et communiquer obligatoirement lors de push, de pull, de nouvelle branches … etc pour que les autres membres ne soient pas laissés dans l'incompréhension.

-Communication de groupe: Parler plus pour gagner plus. Si nous avions clairement exprimé à tous les membres du groupe les mécanismes centraux du jeu, et fait des réunions méthodiques à chaque séance, le projet aurait pu avancer à grands pas.

-Gestion de projet : commencer par le général (pygame, fonction main), pour après et seulement après faire les fonctions précises. Nous avons fait l'inverse, en commençant par implémenter les textures de routes et les vendeurs de nourritures avant d'avoir une démo technique correcte avec les features déjà implémentés correctement. Les objectifs n'étaient pas fixés, or cela nous aurait permis d'avoir des estimations de notre avancée dans le projet, et en l'occurrence de nous rendre compte que nous étions en retard.

-Implication personnelle des membres du groupe en pourcentage (validée par le groupe):

Dollen Quentin	23.5%
Grand Antoine	15%
Gravis Siméon	14%
Le Botlan Pierre Gilles	23.5%
Le Doan Phuoc	20%
Mentou Chelsy	4%