**C41-IN**

**Votre nom : Janesch Bettina-Sarah**

***NOTE : enregistrer le document sous votre numéro de matricule***

|  |
| --- |
| **Instructions pour les Étudiants** |
| 1. **Lecture initiale du code** : Faites une lecture rapide de chaque module afin de voir son rôle a priori. Identifiez le point de démarrage. |
| 1. **Exécution du Code** : Commencez par exécuter le code pour observer son comportement et ses fonctionnalités. |
| 1. **Lecture pour une analyse structurée** : Faites une lecture « procédurale » en avançant dans le code comme le programme le fait. |
| 1. **Identification de l’architecture, des composantes et leurs fonctionnalités** : Notez les fonctionnalités que vous parvenez à identifier et comment elles sont implémentées. |

**Rapport de lecture**

|  |
| --- |
| 1. **Description Générale** : |
| * + But général du programme, développement potentiel dans le cadre du cours. |

Réponse :

But : Ce jeu est une sorte de Age of Empire : 4X – on contrôle un empire et on e**x**plore, on peut e**x**pand l’empire, on e**x**ploite des ressources locales et on e**x**termine des ennemis (les autres joueurs).

Développement potentiel : toute la partie du joueur – créer des bâtiments avec des ressources, interagir avec les autres joueurs et arriver a gagner ou non - donc toute la logique du jeu a faire, car l’environnement est déjà fait.

|  |
| --- |
| 1. **Analyse générale de l’architecture, des composantes et des fonctionnalités** : |
| * + Description de l’organisation du code, préciser le rôle des diverses parties et de leurs interdépendances... |

Réponse :

Controleur et Vue initialisés avant le modele. (Vue : self modele=None car Partie() existe pas encore)

Vue - Charge les frames, les gifs & images …

Controleur depart la mainloop pour la Vue

Créer partie bouton attaché a la methode créer\_partie du Controleur 🡪 vue affiche lobby et maintenant on boucle sur le lobby (controleur)

Bouclage du lobby : ici on initialise la partie qui crée la Partie() du modele

Initialiser partie :modele est faite avec Partie() et la vue a maintenant acces au modele (self.modele) :valeurs initialisés dans plusieurs dictionnaires: des personnages & leur classes, regions, biotopes … et après des méthodes de creation :

* + - Créer\_regions, biotopes,population (avec joueurs)– avec les valeurs des dict de partie
* Vue maintenant fait initialiser\_avec\_modele(), avec les infos de partie()change cadre,centrer\_maison
* finalement : on appelle la **boucle du jeu**

**Vue** : bouton Lancer partie 🡪 contrôleur lancer\_partie : appele le serveur

**Controleur** : **boucler\_sur jeu** : actions & actions\_requises des tab vides au début.

* + On appelle le serveur (try) avec un dictionnaire data qui contient : le nom joueur, a quelle itération on est rendu dans jeu, et le [] d’actions a faire.
  + Si l’appel retourne ATTENTION, on saute le tour (aka le tour a été joué) et self.on joue revient a 0, donc pas a nous de jouer.
  + Si TRY marche : on ajoute les actions a faire dans Partie(modele) avec ‘mondict’ qui a été retourné du serveur (JSON) : le dictionnaire self.actions\_a\_faire de la partie est remplie si la clé dans les actions reçus du controleur existe pas dans le dict du modele.
  + Si try marche pas – erreur : on imprime et on donne le tour au prochain (self.onjoue)

Finalement – **si notre tour** : on joue le prochain coup (modele) avec l’iteration du jeu en param et on affiche (vue)

**Controleur - Jouer\_prochain\_coup :**

- avec iteration ou on est rendu :avec les actions courantes en fonction de l’iteration, on boucle dans le dictionnaire et on attribue les actions au joueur

- deplacement des choses dans le jeu (biotopes)

-demande a chaque joueur de joueur leur prochain coup (rappelle elle-meme)

**Vue - Afficher\_jeu :**

- pour joueur local : montre les ressources & msg general

- affiche les bâtiments du joueur (en construction ou non)

- ajuste les personnages de chaque joueur avec create\_image – chaque perso du dictionnaire du joueur local

- affiches les choses vivantes (du biotope - dict :daim,eau)

- rafraichit le chat

-- retour dans jouer\_prochain\_coup : si pas notre tour : on decremente iteration jeu pour synchroniser avec les autres et maintenant on joue.

A la fin, boucler\_jeu se rappelle avec le delai (qui est 40 a l’init)

|  |
| --- |
| 1. **Stratégie de distribution et d’application des requêtes d’actions des joueurs :** |
| * + Expliquez le principe, et décrivez la syntaxe générale des requêtes. |

Réponse : methode appeler\_serveur du controleur reçoit le URL en param et un dictionnaire qui contient le nom du joueur actuel au minimum (et peut-etre les actions a faire, etc).

-le URL dit au serveur quoi faire, mettons créer une partie

- par defaut, la methode de ‘formulaire’ et GET, et sinon, on précise qu’on veut POST

- la reponse est enregistré dans une variable et contient un objet retourné par le serveur (self.session.get/post)

raise\_for\_status() retourne un objet d’erreur s’il existe

retourne un objet JSON, que nous on met dans un dictionnaire qu’on travaille avec

|  |
| --- |
| 1. **Faire le diagramme de séquence du placement de bâtiments** : |
| * + https://sequencediagram.org/ |

Réponse :joueur faire batiment

