Intelligence Artificielle pour le Jeu Vidéo
Support de cours n°1

Cours introductif

Contact : Agnes Delaborde – agnes.delaborde@gmail.com

Dates
Dimanche 10 avril (minuit) : envoi du code et du rapport (par mail) **Mercredi 13 avril (minuit)** : envoi de la présentation (par mail) **Vendredi 15 avril** : soutenance

Objectifs

Prise en main de l'architecture du code, utilisation du langage Lua et de la librairie Löve

Exercice 1 – A peine commencé, je suis déjà fatigué.

Vous avez lancé la simulation. Vous avez remarqué une chose : Gus fatigue trop vite…

a. Sur l'écran de simulation, repérez : la jauge de vigueur, la valeur de vigueur.
b. Repérez dans les constantes (constants.lua) la variable permettant de diminuer la vitesse à laquelle Gus perd de sa vigueur.
c. Choisissez une valeur rendant la simulation « jouable », c'est-à-dire donnez un temps raisonnable avant que Gus ne s'arrête de marcher. Pas de solution parfaite ici, choisissez juste une valeur que vous trouvez confortable pour tester la simulation. Vous aurez sans doute à adapter cette valeur plus tard.
d. Sur quelle caractéristique de Gus la vigueur a-t-elle un impact direct ?
e. Dans la fonction qui met à jour cette caractéristique, repérez la fonction utilisée pour calculer la nouvelle valeur. Vous pourrez la retrouver dans tools.lua. Quel est le calcul effectué ?

Exercice 2 – Tout est sous contrôle.

Gus sait maintenant mieux gérer sa fatigue physique.

a. Dans main.lua : recherchez la méthode love permettant d'intercepter les entrées clavier.
b. Quelles sont les touches permettant de basculer entre les modes de déplacement de Gus (aléatoire, ou contrôle clavier) ?
c. Dans gus.lua : quelle est la fonction gérant le déplacement aléatoire de Gus ? Et son déplacement au clavier ?
d. Dans le mode de déplacement aléatoire, à quoi sert la variable desireToTurn ?
e. Faites en sorte que Gus ait envie de changer de direction plus souvent, lorsqu'il se déplace aléatoirement. Il s'agit juste de changer la valeur d'une variable, ne cassez pas tout !

Exercice 3 – Dessine-moi un mouton.

A **gauche** de la jauge de vigueur, nous allons créer une seconde jauge, afin d'afficher le niveau de sociabilité de Gus.

a. Dans constants.lua : repérez les variables ayant trait aux niveaux maximum et minimum de sociabilité, ainsi que la variable représentant le pas lors d'un changement du niveau de sociabilité.

b. Dans main.lua, méthode love.load() :

    1. Dupliquez la partie de code permettant de déclarer une instance de Gauge :

        – Vous appellerez cette jauge « gauge_sociability ».

        – Modifiez le nom du fichier image contenant le caption (Gauge_Sociability_Caption.png dans votre dossier assets/)

        – Modifiez le label (vous trouverez sa valeur dans constants.lua)

        – Modifiez les positions du container et du caption sur l'axe y (on place la jauge sociabilité à gauche de la jauge vigueur).

    2. Créez les objets Image associés à cette dimension (un objet pour le « container » et un pour le « caption »), en vous inspirant du code écrit pour la jauge de vigueur. La méthode utilisée est love.graphics.newImage().

c. Dans main.lua, méthode love.draw() :

    1. Dupliquez et adaptez la partie de code permettant l'affichage de cette jauge.

    2. Allez regarder le code de la méthode draw (classe Gauge), dans gauges.lua. Cette méthode fait appel à la méthode love.graphics.draw.

    3. Assurez-vous que la hauteur de la jauge dépend de la variable Gus.sociability. Vous trouverez les valeurs min et max dans constants.lua.

    4. Arrivés à ce point, méditez sur le fait que la clarté, la lisibilité, et la richesse des commentaires de votre code compteront pour 3 points de la note finale.

d. Maintenant que votre nouvelle jauge s'affiche sans encombre à l'écran, vous constatez que par défaut, la valeur de sociabilité de Gus est de 40. Modifiez le code (dans gus.lua) afin que Gus commence avec la valeur maximum de sociabilité (définie dans constants.lua).

Exercice 4 – Gus se fait des amis

Gus n'est pas seul dans la pièce. Vous avez certainement tenté d'aller interagir avec les quatre PNJ, sans succès. Même la belle blonde n'était pas intéressée par le charme ravageur de Gus.

Avant de rendre les PNJ plus interactifs, étudions-les :

a. Dans constants.lua, trouvez le tableau NPC_LIST. Regardez les profils des PNJ : arrogant, charmeur, agressif, timide. D'après les PNJ que vous voyez à l'écran, identifiez le profil de chaque PNJ (d'après ses coordonnées ou son allure physique).

b. Choisissez un nouveau nom pour chacun des PNJ. Vous voyez dans la liste NPC_NAMES qu'ils se nomment « NPC1 », « NPC2 », « NPC3 », et « NPC4 ». Remplacez ces noms par des noms plus « naturels ».

c. Dans main.lua, recherchez les appels à la fonction createNPC. Ici sont crées les instances de la classe NPC. Reportez vos nouveaux noms ici également : NPC1 = createNPC("nouveau nom"), etc. Ne modifiez pas les noms des variables.

d. Rendons les PNJ interactifs maintenant. Trouvez le label ::pass_npc:: dans gus.lua. Supprimez le label, ainsi que l'appel à goto pass_npc. Exécutez le code, et allez dialoguer avec un PNJ :

    1. Le dialogue s'affiche à droite de l'écran. Les noms de vos PNJ doivent apparaître au cours du dialogue (sinon, c'est l'heure de la phase debug).

    2. Gus n'arrive plus à s'éloigner du PNJ, n'est-ce pas ?

        – Trouvez le raccourci clavier permettant de mettre un terme à l'interaction (en étudiant le code, pas en essayant au hasard des touches au clavier…).

        – Sur quel booléen ce raccourci agit-il ?

        – Dans quelle partie du code voit-on que ce booléen peut empêcher Gus de se déplacer ?

– Dans quelles circonstances le booléen repasse-t-il à Vrai ?

3. Maintenant que vous savez décoincer Gus, allez lancer le dialogue avec chaque PNJ.
4. Si vous retournez voir un PNJ déjà rencontré, vous constatez que le dialogue ne se relance pas. Etudiez le code pour déterminer pourquoi.

Exercice 5 – Gus n'a plus d'amis

Chaque PNJ tient le même discours. Nous avons pourtant vu que les PNJ semblent être dotés de profils différents : nous imaginons donc qu'ils devraient interagir différemment avec Gus. Nous allons maintenant fournir la possibilité au PNJ de refuser de donner son nom.

a. Allez étudier le dialogue vu dans l'exercice précédent. L'automate se trouve dans dialogues/ask_for_name.lua. L'automate est excessivement simple (3 états parcourus successivement, pas de boucles).

b. Quel est l'état dans lequel Gus stocke une information dans sa mémoire ? Quelle est cette information, et par quel moyen Gus en a-t-il pris connaissance ?

c. Dans state_Gus2, lorsque Gus dit « Enchanté, *x*, moi c'est Gus. », le nom du PNJ est-il obtenu en requêtant sur le label de l'object NPC manipulé pendant ce dialogue, ou bien en interrogeant la mémoire de Gus ? Saisissez-vous l'importance de différencier ces deux sources d'informations (la « mémoire du programme » et la « mémoire de Gus ») ?

d. Vous allez modifier le dialogue afin de pouvoir gérer un éventuel refus de la part du PNJ de donner son nom. Suivez ces consignes :

   1. Utilisez la méthode isWillingToProvidePersonalData() de la classe NPC. Adaptez la conditionnelle dans cette méthode selon vos propres critères si vous l'estimez incorrecte (inutile d'aller trop loin dans la réflexion pour le moment, jouez juste avec les étiquettes de profil des PNJ).
   2. Les phrases de dialogue sont stockées dans data/dialogues.lua. Si vous avez besoin de nouvelles phrases, ajoutez-les à cet endroit.
   3. Si le PNJ refuse de donner son nom, Gus lui dira simplement au revoir (on ne codera pas de stratégies pour l'instant).

Exercice 6 – Tout en nuance

Nous souhaitons que toutes les phrases prononcées par le PNJ soient « typées » par la personnalité de celui-ci. Jusque là, vous avez deux phrases possibles pour les PNJ. Soit (Cas A) « Je m'appelle %s. », soit (Cas B) « Je refuse de te donner mon nom ».

a. Pour chaque PNJ, ajoutez dans data/dialogues.lua une phrase pour Cas A et Cas B. Par exemple, le timide pourra dire « Euh non… Désolé… Je peux pas vous parler là tout de suite... », et le charmeur « Ah non mon p'tit loup, t'es mignon mais je te donnerai pas mon nom ! ». Prenez en compte ces remarques :

   1. Vous aurez peut-être besoin de changer la structure des données dans le fichier data/dialogues.lua : des tableaux associatifs plutôt que des strings. C'est votre code maintenant, faites vos propres choix.
   2. Créez bien une phrase pour tous les cas (même si dans notre scénario actuel certains PNJ vont refuser et d'autres accepter), nous aurons certainement besoin de toutes les phrases plus tard.
   3. A cette occasion, commencez à réfléchir aux différentes caractéristiques qui pourront plus tard vous permettre de discriminer automatiquement le profil du PNJ à partir de son style langagier (longueur des phrases et des mots, vocabulaire utilisé, marqueurs particuliers, etc.). Si vous réussissez à établir quelques règles, appliquez-les quand vous inventez vos phrases. Sinon, nous y reviendrons de toute façon dans les séances à venir.
   4. Pas de censure quant au choix des énonciations. Plus votre PNJ sera caricatural, plus il vous sera facile, dans la suite du projet, de détecter automatiquement son profil.

b. Modifiez le script afin que la phrase prononcée soit sélectionnée en fonction du profil du PNJ. Sachant que nous créerons plus tard d'autres dialogues nécessitant que le PNJ s'exprime en fonction de sa personnalité, ne faites pas du code ad hoc juste pour ce dialogue.

Projet – Propositions relatives à ce cours

Proposition 1 : Au chargement du jeu, les PNJ apparaissent à des endroits aléatoires de la carte.

Proposition 2 : Le PNJ met un peu de temps avant de répondre à Gus, afin de simuler un « temps de réflexion ».

Proposition 3 : Le PNJ peut mentir sur son nom. Si les PNJ sont dotés de la capacité à mentir :

→ Implication 1 : soit Gus croira sur parole et intégrera ce nom à sa mémoire (cas le plus simple),

→ Implication 2 : soit Gus tentera de vérifier les informations données pendant le dialogue (plus complexe, à envisager ultérieurement).