Plan de tests

Projet L3D2 - Systèmes proies-prédateurs

Année 2022-2023

***Les informations d’identification du document***

| Référence du document: | PT\_L3D2\_v3 |
| --- | --- |
| Version du document: | 3 |
| Date du document: | 14/05/2023 |
| Auteurs: | Fu Suntanqing  Rossignol Paul  Xu Remy  Ye Frédéric |

***Les éléments de vérification du document***

| En attente de validation: | LATTAUD Claude |
| --- | --- |
| Soumis le: | 19/04/23 |
| Validé le: | \_\_/\_\_/23 |
| Type de diffusion: | Document électronique (.pdf) |
| Confidentialité: | Réservé au jury, professeurs encadrants et aux étudiants de l’Université Paris Cité. |

**SOMMAIRE**

[**I. Introduction 4**](#_1s4ci3dztqq)

[1) Objectifs et méthodes 4](#_ci8tadddealr)

[2) Documents de référence 4](#_ld3n3mwq4h9y)

[3) Conformité aux spécifications générales 4](#_xtc6ldn6d1sz)

[**II. Tests fonctionnels 4**](#_y8roi79eifye)

[1) Ouverture de l’application et sélection des nouvelles espèces ajoutées 4](#_3w6nrojhggjc)

[2) Prise de contrôle d’un agent et mimétisme 5](#_2ut2u5hgbnrh)

[3) Amélioration de l’intelligence artificielle des agents 8](#_z2wr9a84amav)

[4) Système de météo: 9](#_y8ljoh5bmo4y)

[5) Vérification du système de caractère d’agent : 12](#_awmq22rfz3mh)

[6) Fonctionnalités supplémentaires à ceux définies dans la phase de spécifications. 12](#_38z20hknz3di)

[**III. Tests unitaires 15**](#_yd73qp9z2op6)

[1) Contrôle d’agent 15](#_hxz9vhd6qm4m)

[2) Interface graphique 18](#_4zxmccdrc6gy)

[3) Système de meute 19](#_3lovbfr4r0ca)

[4) Système de mimétisme 22](#_d2y3lvquaq2o)

[5) Téléportation au-dessus d’un agent 24](#_u3t8bh4xxdw8)

[6)Suppression d’agent 26](#_h28atx5tx2g1)

[7)Système de peur 26](#_8lnnnbb5pnvd)

[8) Sauvegarde et chargement d’une simulation passé 27](#_j6ernyg0c2cn)

[9)Fonction concernant l’interface utilisateur 27](#_5lrgmq83rwty)

[**IV. Bibliographie 28**](#_ckfcx16mrvg1)

# I. Introduction

## 1) Objectifs et méthodes

Ce document aura pour but de montrer le bon fonctionnement général de notre logiciel de trois manières distinctes :

* Vérifier si les tests des différentes fonctionnalités établies dans le cahier des charges et les tests de la conformité aux spécifications générales du cahier de recette fonctionnent (tests fonctionnels).
* Vérifier si les différents modules, paquetages,... de l’application s’interfacent correctement (tests d’intégration).
* Et enfin vérifier si les différentes méthodes que nous avons implémentées cette année fonctionnent de manière attendue comme pour la conception détaillée. (tests unitaires).

## 2) Documents de référence

Les trois documents de références sont: Notre cahier des charges (CC\_L3D2), notre cahier de recette (CRec\_L3D2) et notre conception détaillée (CD\_L3D2) (ou bien plan de développement PD\_L3D2).

## 3) Conformité aux spécifications générales

Pour vérifier que l’application fonctionne comme nous le souhaitons nous effectuerons les tests sur l’ensemble des scénarios suivants, il est important de vérifier ces scénarios car l’échec d’un scénario pourrait remettre en cause l’utilité de notre application, c’est pourquoi nous devons nous assurer que l’ensemble de ces scénarios de tests sont effectué avec succès.

# II. Tests fonctionnels

Dans cette partie, nous corrigeons et ajoutons plus de précisions sur les tests du cahier de recette et nous ajoutons les fonctionnalités optionnelles que nous avons ajoutées durant la phase de développement.

## 1) Ouverture de l’application et sélection des nouvelles espèces ajoutées

* **Id 1.1** Ouverture de l’application :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le .exe | Avoir un système d’exploitation Windows. | L’application doit se lancer avec le menu principal. | L’application s’ouvre bien: le test est validé. |

* **Id 1.2** Sélection des nouvelles espèces ajoutées :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix.  3-Modifier le nombre d'entités d’une nouvelle espèce dans la simulation.  4- Lancer la simulation. | Avoir ajouté les nouvelles espèces dans les paramètres et lancer la simulation. | L’ensemble des nouvelles espèces doivent normalement apparaître dans la simulation et avoir des comportements correspondants à la réalité. | Toutes les espèces sont bien présentes et peuvent être ajoutées, le test est donc validé. |

* **Id 1.3** Ajout d’un ou plusieurs agents pendant la simulation :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Choisir un terrain au choix.  3- Lancer la simulation.  4- Appuyer sur les icônes proie, agent ou autotrophe.  5- Ajouter un ou plusieurs agent(s) de l’espèce sélectionnée avec les caractéristiques au choix ou celles prédéfinies.  6- Attendre que la transition entre la vue normale et la vue ajout d’agent.  7- Cliquer n fois (n nombre d’agent à ajouter) sur l’endroit où l’on veut faire apparaître les agents. | Avoir lancé la simulation, et avoir ajouté les espèces comme à la procédure de test. | L’ensemble des nouvelles espèces doivent normalement apparaître dans la simulation et avoir des comportements correspondants à la réalité. | Si les nouvelles espèces sont bien présentes dans la simulation alors le test est réussi. |

## 2) Prise de contrôle d’un agent et mimétisme

* **Id 2.1** Contrôle d’un agent avec le déplacement, le saut et le sprint:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix.  3- Lancer la simulation.  4-Cliquer sur un agent qui n’est pas un autotrophe.  5-Sélectionner l’option “Contrôle manuel de l’agent”.  6- Déplacer et effectuer des actions avec le clavier. | Avoir lancé la simulation et que des agents\*contrôlables y soient présents. | Lorsque l’on clique sur l’agent l’option “prendre le contrôle” est disponible et une fois sélectionné, l’agent s’arrête de bouger. Ensuite, nous pouvons déplacer l’agent avec les flèches directionnelles, le faire sauter avec la touche “J” et le faire sprinter avec la touche “Ctrl” gauche. | L’agent est contrôlable, il peut sauter et accélérer. Ainsi le test est validé . |

**\*agent : Animal ou plante ajouté dans la simulation qui a un comportement**

**/!\ les plantes ne sont pas contrôlables**

* **Id 2.2** Réalisation de l’action manuelle boire :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix.  3- Lancer la simulation.  4-Cliquer sur un agent contrôlable.  5-Sélectionner l’option “prendre le contrôle”.  6-Les boutons/toggle boire, manger et dormir sont activés  7-Déplacer l’agent jusqu’à la source d’eau.  8- Cliquer sur le bouton boire | Avoir lancé la simulation et que des agents contrôlables y soient présents. Il faut ensuite bien déplacer l’agent à une source d’eau. | Lorsque l’on clique sur l’agent, puis sur l’option “prendre le contrôle” puis sur le bouton boire, la barre de soif doit se mettre à 0. L’action devient DrinkAgentAction pendant l’action (on ne le voit pas car boire est instantané). | La barre de soif de l’agent diminue bien à 0. Ainsi le test est validé. |

* **Id 2.3** Réalisation de l’action manuelle manger :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix.  3- Lancer la simulation.  4-Cliquer sur un agent contrôlable.  5-Sélectionner l’option “prendre le contrôle”.  6-Les boutons/toggle boire, manger et dormir sont activés  7-Déplacer l’agent jusqu’à une de ses proies  8- Cocher la case manger | Avoir lancé la simulation, que des agents contrôlables y soient présents et qu’au moins une des proies de l’agent est bien dans la simulation. | Lorsque l’on clique sur l’agent l’option “prendre le contrôle” puis en cochant le bouton manger, il doit attaquer sa/ses proies dès qu’il est à portée d’attaque. L’action devient FindFoodAction. | L’agent tue une de ces proies qui se trouve à proximité et le mange. Ainsi sa barre de faim diminue. |

* **Id 2.4** Réalisation de l’action manuelle dormir :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix.  3- Lancer la simulation.  4-Cliquer sur un agent contrôlable.  5-Sélectionner l’option “prendre le contrôle”.  6-Les boutons/toggle boire, manger et dormir sont activés  7- Cocher la case dormir. | Avoir lancé la simulation, que des agents contrôlables y soient présents et que la barre d'endurance soit à moins de 100%. | Lorsque l’on clique sur l’agent, puis l’option “prendre le contrôle”, puis sur dormir, nous ne pouvons plus réaliser des déplacements, ni boire et ni manger manuellement. La barre d'endurance se remplit tant que la case est cochée. Lorsque l’agent est à 100% d'endurance, cela désactive dormir manuellement. L’action devient SleepAgentAction. | L’agent dort correctement et remonte sa barre d’endurance. |

* **Id 2.5** Nage manuelle dans l’eau :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix  de données.  3-Lancer la simulation.  4-Prendre le contrôle manuel d’un agent  5- Le déplacer dans l’eau | Lancer la simulation. L’agent se trouve près d’une source d’eau. | L’agent doit être ralenti une fois dans l’eau. Le sprint dans l’eau est aussi diminué (1.5 fois la vitesse dans l’eau). L’agent doit flotter à la surface du point d’eau. | L’agent nage correctement comme dans “Résultat prévu” le test est donc validé. |

## 

* **Id 2.6** Fonctionnalité de mimétisme au sein d’une espèce:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation aux choix.  3- Lancer la simulation.  4-Cliquer sur un agent contrôlable.  5-Sélectionner l’option “prendre le contrôle”.  6- Déplacer et effectuer des actions avec le clavier. | Lancer la simulation. Deux agents de la même espèce sont dans la même zone d'action et ont les mêmes besoins (faim, soif, contourner un obstacle…).  L’agent copié peut être contrôlé par l’utilisateur effectuant les actions suivantes: boire, manger, fuir, dormir. L’agent copieur reconnaît une action qui vérifie les conditions nécessaires à la copie. | Une fois l’action effectuée par l'humain ou non, l’agent reproduira cette action si elle vérifie les conditions nécessaires à la copie. | L’agent copie l’action qu’il a observé si les conditions nécessaires à la copie sont satisfaites. Ainsi le test est validé. |

## 3) Amélioration de l’intelligence artificielle des agents

* **Id 3.1** Système de meute:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix  de données.  3-Lancer la simulation.  4-Créer un mâle et une femelle d’une même espèce et attendre un accouchement. | Lancer la simulation, créer un mâle et une femelle d’une espèce et qu’ils fassent un enfant ensemble. | Les agents de la meute se déplacent désormais ensemble. Un alpha est choisi en fonction de ses caractéristiques. | La meute est bien créée et un alpha est bien désigné ainsi le test est validé. |

## 

* **Id 3.2** Système de fatigue pendant et après la nage :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix  de données.  3-Lancer la simulation.  4-Observer un agent qui nage | Lancer la simulation. L’agent se trouve près d’une source d’eau. | L’agent se fatigue progressivement: Il y a 5 étapes de fatigue et à chaque étape il est un peu plus ralenti. Une fois à la surface, l’agent se rétablit de la fatigue progressivement. | L’agent se retrouve ralenti dans l’eau, et également une fois qu’il rejoint la surface (chaque palier de fatigue dure 400 secondes). Il se rétablit 10 fois plus rapidement qu’il ne se fatigue. Ainsi le test est validé |

* **Id 3.3** Système d'oxygène:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1-Ouvrir le laboratoire.  2-Changer les paramètres de la simulation au choix  de données.  3-Lancer la simulation.  4-Observer un agent qui nage | Lancer la simulation. L’agent se trouve près d’une source d’eau. | Le niveau d'oxygène de l’agent se consomme progressivement lorsque l’agent se trouve dans l’eau. Si l’agent n’a plus d'oxygène, il meurt.  Une fois retourné à la surface, il retrouve de l'oxygène progressivement. | L’agent meurt au bout d’un certain temps de manque d'oxygène (5 minutes de simulation). L’agent se rétablit 10 fois plus rapidement qu’il ne perd de l'oxygène. Ainsi le test est validé. |

* **Id 3.4** Système de peur :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation. L’agent doit se retrouver soit assez près d’un de ses prédateurs, soit l’agent doit voir un animal de son espèce se faire chasser, soit il y a de l’orage. | Lorsqu’un agent est face à une situation qui lui fait peur, sa jauge de peur augmente. Au-delà d’un certain seuil moyen, l’agent s’enfuit dans une direction opposée à son prédateur s’il y en a un, lorsque la jauge de peur est au maximum l’agent est terrifié et est immobilisé. | Lorsque l’agent se trouve à 10% de peur ou plus, l’agent s'enfuit dans une direction opposée à celle de son prédateur s’il y en a un et lorsque la barre de peur sera au maximum, l’agent sera immobilisé validant ainsi le test. |

## 4) Système de météo:

* **Id 4.1** Vérification du système de saison :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre qu’un changement de climat ait lieu. | Le panneau d’information en haut à gauche affiche bien la nouvelle saison lors du changement, ce qui entraîne le choix d’une nouvelle météo en fonction de la saison. | Le test est validé. |

## 

* **Id 4.2** Vérification du système de sécheresse :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre qu’une sécheresse ait lieu ou la créer manuellement. | Conduit à une baisse de la reproduction des plantes et pour les animaux à une baisse plus rapide de leur barre de soif et le niveau de température augmente. | Le test du système de sécheresse est validé. |

* **Id 4.3** Vérification du système de pluie :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre qu’une pluie ait lieu ou la créer manuellement. | Affichage dans le panneau d’information que l’état de la météo est “Rain” et conduit à une modification logique des statistiques des agents et de la température. | Le test est une réussite. |

* **Id 4.4** Vérification du système de Vent:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre qu’un vent ait lieu ou la créer manuellement. | Variation de la température et les agents dépensent plus d’endurance. | Le test est validé. |

### 

* **Id 4.5** Vérification du système d’orage :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre que l'orage ait lieu ou la créer manuellement. | Augmentation de la peur des agents et variation de la température. | Le test est validé. |

* **Id 4.6** Vérification du système de neige :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre que la neige ait lieu ou la créer manuellement. | Diminution de la consommation d'eau, changement significatif de la température et apparition de particules de neige. | Le résultat prévu est validé. |

* **Id 4.7** Vérification du système de tempête :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre que la tempête ait lieu ou la créer manuellement. | Augmentation du taux de consommation d’endurance des agents et variation de la température. | Le test est validé. |

* **Id 4.8** Vérification du système de temps ensoleillé(clair):

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation. | Lancer la simulation et attendre que le temps ensoleillé (clair) ait lieu ou la créer manuellement. | Le panneau affiche bien que le temps actuel est clair, cette météo est considérée comme l’état normal du monde ainsi il n’y a aucune modification de statistique pour les agents. | Le test est validé. |

## 5) Vérification du système de caractère d’agent :

* **Id 5.1** Vérification du système de caractère d’agent :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Changer les paramètres de la simulation au choix.  3- Lancer la simulation.  4- Créer un ou plusieurs agents | Lancer la simulation, et des agents doivent exister dans la simulation. | L’agent possède un trait parmi les suivants: Peureux, solitaire, courageux, social, … | L’agent se voit bien attribuer un trait de caractère, ainsi le test est validé. |

## 6) Fonctionnalités supplémentaires à ceux définies dans la phase de spécifications.

* **Id 6.1** Système de téléportation sur les agents:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Lancer la simulation en choisissant le terrain de base.  3- Ajouter un ou plusieurs agents dans la simulation.  4- Appuyer sur la touche “T” ou appuyer sur une espèce particulière. | Lancer la simulation et si on utilise la touche “T” la présence au moins d’un agent et pour une espèce particulière l'existence d’au moins un agent de l’espèce sélectionnée. | La caméra principale se place au-dessus d’un agent. | La caméra passe bien vers un agent ainsi le résultat prévu est validé. |

Note: Nous nous téléportons sur les agents par leur ordre de création. Maintenir la touche “T” permet de suivre un agent.

* **Id 6.2** Système de combat entre les alphas:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Lancer la simulation en choisissant le terrain de base.  3- Ajouter un couple mâle/femelle de l’espèce loup (sans le trait de caractère solitaire).  4- Attendre qu’une meute se forme avec un accouplement.  5- Monter les valeurs de faim ou de soif des agents de la meute à 80%. | Lancer la simulation et présence d’au moins un couple (sans le trait de caractère solitaire) d’une même espèce et d’au moins un agent de la même espèce. La meute doit être en mauvais état et qu'un combat ne s’est pas produit récemment (1000 tics dans la simulation). | L’action de l’alpa et du loup désigné sera établie à “Fight”  et ils se chercheront afin de provoquer un combat entre les deux agents. | L’action des deux agents est bien modifiée en combat, le test est validé. |

* **Id 6.3** Suppression d’un agent:

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Lancer la simulation en choisissant le terrain de base.  3- Ajouter un agent dans la simulation.  4- Cliquer sur l’agent pour faire afficher la fenêtre de statistique d’agent.  5- Cliquer sur le bouton “+” de la fenêtre graphique pour faire apparaître une deuxième fenêtre.  6- Cliquer sur le bouton “Supprimer l’agent” pour supprimer l’agent. | Lancer la simulation et présence d’au moins un agent. | L’agent disparaît de la simulation. | L’agent est bien supprimé ainsi le test est validé. |

* **Id 6.4** Chasse en meute :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Lancer la simulation en choisissant le terrain de base.  3- Ajouter un couple mâle/femelle d’une espèce (sans le trait de caractère solitaire).  4- Attendre qu’une meute se forme avec un accouplement.  5- Un agent a sa barre de faim à 50%  6- Placer un nouvel agent qui est listé dans la liste des proies des agents qui sont en meute. | Lancer la simulation et présence d’au moins un couple (sans le trait de caractère solitaire) et présence d'une proie de l'espèce de la meute . | L’agent se déplace vers la proie et l’attaque, tous les agents de la meute qui cherchent une proie se voient attribuer cette proie. | L’ensemble des agents qui veulent trouver une proie se voit attribuer la proie. Si c’est le cas, le test est une réussite. |

* **Id 6.5** Laisse l’Alpha du groupe manger en premier :

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Lancer la simulation en choisissant le terrain de base.  3- Ajouter un couple mâle/femelle d’une espèce (sans le trait de caractère solitaire).  4- Attendre qu’une meute se forme avec un accouplement.  5- Qu’un agent a sa barre de faim à 50%.  6- Placer un nouvel agent qui est listé dans la liste des proies des agents qui sont en meute. | Lancer la simulation et présence d’au moins un couple (sans le trait de caractère solitaire) et avoir une proie morte. | Une fois la proie morte, si l’alpha cherche une proie, il attend que celui-ci mange avant de pouvoir lui-même manger. | Le test est une réussite, l’alpha mange en premier. |

* **Id 6.6** Sauvegarde des données d’un agent dans un fichier

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Lancer la simulation en choisissant le terrain de base.  3- Ajouter un agent dans la simulation.  4- Appuyer sur le bouton Échap pour ouvrir le menu de pause  5- Appuyer sur le bouton Enregistrer la simulation | Lancer la simulation et présence d’au moins un agent. | Les données de chaque agent sont enregistrées dans un fichier dans le dossier “SaveFolder”. | Les données sont bien sauvegardées dans le fichier, le test est donc une réussite. |

* **Id 6.7** Chargement des données d’un agent à partir d’un fichier

| Procédure de test | Dépendance | Résultat prévu | Résultat du test |
| --- | --- | --- | --- |
| 1- Ouvrir le laboratoire.  2- Appuyer sur Lancer la simulation  3- Appuyer sur Lancer une ancienne simulation | Lancer la simulation et présence de fichier contenant les données des agents d’une simulation précédente | Les agents dont les données ont été enregistrées  sont recréés dans la simulation. | Les anciens agents apparaissent correctement ainsi le test est une réussite. |

# III. Tests unitaires

Afin d’effectuer les tests unitaires sous unity, il existe l’outil Test runner inclus dans le logiciel Unity (Window ->General ->TestRunner). Cependant, le projet est un projet sous plusieurs années et que les tests unitaires n’ont pas été réalisés les autres années. Certains “Assembly definitions” ne peuvent pas être définis car certaines dépendances sont cycliques (par exemple DBHelper et Agent). Il faudrait fusionner des classes et réarranger la structure du projet.

Nous avons donc choisis de décrire les tests de la même manière que précédemment.

De plus, en raison de notre cycle de vie en modèle V comme indiqué dans le plan de développement, il se peut que nous n’avions pas prévu certaines méthodes et attributs avant la phase de développement.

Dans la suite, les tests unitaires n'incluent pas les méthodes Update(), Awake(), Start(), etc. (À part s’ils ont une grande partie de code.) Nous nous contenterons d’explications sur les actions qu’ils effectuent et/ou les ajouts majeurs que l’on a apportés.

## 1) Contrôle d’agent

1. Tests des fonctions de ManualControl:

* **Id1.1.1** void ApplyGravity(bool grounded):

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de modifier la vélocité en y de l’agent en fonction de s’il est au sol ou pas. | Aucune dépendance avec d’autres classes pour ce test unitaire. | Vérifier si grounded est bien modifié dans la classe Agent dans les méthodes OnCollisionEnter et OnCollisionExit. | grounded est un booléen permettant de savoir si l’agent est au sol ou en l’air.  S’il est au sol, la méthode modifie la vélocité en y en une valeur faible pour ne pas le ralentir. (-1.0f)  Sinon, la méthode modifie la vélocité en y élevant dès qu’il est en l'air. |

* **Id1.1.2** GameObject FindChildGameObjectByName(GameObject rootParentGameObject, string gameObjectName):

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de chercher un GameObject fils à n’importe quel degré avec son nom. | Aucune dépendance avec d’autres classes pour ce test unitaire | Vérifier si “rootParentGameObject” existe bien. | “rootParentGameObject” est le GameObject père dont on veut chercher ses fils et ses petits fils etc. “gameObjectName” est le nom du fils qu’on veut chercher.  La fonction doit soit retourner le GameObject associé à “gameObjectName”, ou bien à null s’il n’existe pas de GameObject à ce nom. |

* **Id1.1.3** float relativeAgentSpeed(Agent agent):

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de donner la vitesse de l’agent pour le contrôle manuel. Le panda après conversion ne peut pas se déplacer donc on donne une valeur fixe. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini pour que la méthode fonctionne bien. | “agent” est l’agent dont on veut accéder la vitesse. La méthode convertit la vitesse d’un String à un float et la retourne. Cas particulier pour le panda où l’on retourne directement une vitesse de 3f. |

1. Tests des fonctions des actions manuelles des scripts EatToggle, SleepToggle, et DrinkButton:

* **Id1.2.1** void chasserManuellement():

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de faire attaquer et manger dès qu’une proie de l’agent est dans un rayon proche. Le déplacement doit être fait par l’utilisateur. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini, et que le contrôle manuel de ManualControl fonctionne pour faire déplacer l’agent. | Dès que la proie de l’agent est à une certaine distance de l’agent, l’attaque et le mange. |

* **Id1.2.2** Agent ProieLaPlusProche(Agent agent)

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de déterminer quelle est la proie la plus proche de celles en vue de l’agent. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini, qu’il y a bien des agents en vue de l’agent. Il faudra vérifier si la méthode animauxDansFov() dans le script agent fonctionne bien. | Parmi tous les agents qu’ils voient à cet instant, retourne la proie la plus proche parmi ceux qu’il voit. |

* **Id1.2.3** voidManualEatAction()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de changer le nom de l’action en cours par FindFoodAgentAction. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et si le GameObject EatToggle fonctionne correctement et que EatToggle a son OnClick() configuré avec cette méthode. | Lorsque EatToggle devient vrai, on change l’action en FindFoodAgentAction, sinon on lui rétablit l’action avant de passer en chasse manuel. |

* **Id1.3.1** voidManualSleepAction()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de changer le nom de l’action en cours par SleepAgentAction. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et si le GameObject SleepToggle fonctionne correctement et que SleepToggle a son OnClick() configuré avec cette méthode. | Lorsque SleepToggle devient vrai, on change l’action en SleepAgentAction, sinon on lui rétablit l’action avant de passer en chasse manuelle. |

* **Id1.4.1** void DrinkButtonOnClick()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de faire boire l’agent dès qu’il est suffisamment proche d’une source d’eau | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini, si le GameObject pointEau existe bien et si le GameObject DrinkButton est bien configuré et qu’il renvoie cette fonction avec OnClick() | Faire boire l’agent en repassant sa barre de soif à 0. |

* **Id1.4.2** voidManualDrinkActionEnter()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de changer le nom de l’action en cours par DrinkAgentAction. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et que DrinkButtonOnClick() contenant cette méthode-là fonctionne bien. | Change l’action en DrinkAgentAction tout en stockant l’action précédente. |

* **Id1.4.3** voidManualDrinkActionExit()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de rétablir l’action avant de boire manuellement. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et que DrinkButtonOnClick() contenant cette méthode-là fonctionne bien et ManualDrinkActionEnter() est bien appelé. | Rétablit l’action avant de boire manuellement. |

## 2) Interface graphique

1. Tests des fonctions de AgentStatsGUI:

* **Id2.1.1** void OnExitButtonClick():

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de désactiver le mode manuel et les actions manuelles. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et les boutons d’actions manuelles existent aussi.  Le bouton dans AgentStatsPanel->TitleBar->Button doit avoir son OnClick bien défini dans cette fonction. | Décoche les actions manuelles et les désactive. Ferme le panel AgentStatsPanel. |

1. Tests des fonctions de AgentStatsGUIUpdater:

**Id2.1.2** void UpdateGUI():

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’update le panel AgentInfoSupp | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et que le panel AgentStatsPanel soit actif. | Appuyer sur le bouton + qui se trouve sur le panel AgentStatsPanel et vérifier que les informations sont bien à jour. |

## 3) Système de meute

1. Test des fonctions d' Agent

* **Id3.1.1** void GestionPack():

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de gérer les actions dans la meute | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini. | Lancer la simulation et ajouter un agent. Si notre agent possède une meute, alors il appelle les fonctions dans GestionPack() |

* **Id3.1.2** void foundNewAgentForPack():

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’ajouter des agents dans la meute en fonction de la vision | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini, qu’il possède une meute. | Lancer la simulation et ajouter un agent. L’agent doit alors s'il a une meute et que cette meute a une évaluation positive, ajouter l’agent de la même espèce qui n’a pas de meute. |

* **Id3.1.3** void FightForTheAlphaPlace():

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet à deux agents d’une même meute de se désigner mutuellement comme ennemi. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini, qu’il possède une meute. | Lancer la simulation et ajouter un agent. L’agent doit alors s'il a une meute et que cette meute a une évaluation  négative changeant d’action en Fight. |

* **Id3.1.4** void createPack(GameObject b):

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de créer une meute avec un autre gameObject b | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini | Lancer la simulation et ajouter un agent. S’il se reproduit avec un agent sans meute alors une meute doit être créée si les conditions sont remplies (pas le caractère solitaire et est une espèce qui peut créer un groupe). |

* **Id3.1.5** double EvalPack(GameObject b):

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’évaluer une meute. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’il possède une meute. | Lancer la simulation et ajouter un agent. Si la meute a faim ou soif retourne -1 si elle est positive retourne 1 sinon retourne 0. |

* **Id3.1.6** void AddAgentToPack(GameObject a)

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’ajouter un agent a une meute. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’il possède une meute. | Lancer la simulation et ajouter un agent. Qu’une des conditions soit remplie pour ajouter des agents et qu’il soit ajouté dans la meute. |

* **Id3.1.7** void DesigneAlpha()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de désigner un alpha dans la meute | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’il possède une meute. | Lancer la simulation et ajouter un agent. Qu’il crée une meute, alors un alpha doit être désigné. |

* **Id3.1.8** void setTypeAgentForPack()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de modifier le typeAgent de l’agent | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini. | Lancer la simulation et ajouter un agent. Il doit avoir le typeAgent WithoutPack et s'il a une meute WithPack ou Alpha. |

1. Classe Fight

-**Id3.2.1** void fight()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet à deux agents de se combattre. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’il a un agent ennemi. | Lancer la simulation et ajouter un agent. Il doit posséder une meute, attendre que la meute soit dans un état critique, les agents vont alors avoir l’action fight et vont se battre jusqu'à la mort d’un des deux agents |

**Id3.2.2** void fight()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de retourner un booléen si l’agent vainqueur doit épargner ou non l’agent adverse. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’il a un agent ennemi. | Lancer la simulation et ajouter un agent. Il doit posséder une meute, attendre que la meute soit dans un état critique, les agents vont alors avoir l’action fight et vont se battre, avant le dernier coup fatal l’agent doit choisir s'il épargne ou non son adversaire. |

## 4) Système de mimétisme

1. Test des fonctions d’agent

* **Id4.1.1** bool PredateurVisible()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de savoir si un prédateur de l’agent se trouve dans le champ de vision de l’agent. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’un prédateur soit en vue. | Lancer la simulation et ajouter un agent. Si un prédateur est en vue, la fonction retourne Vrai |

* **Id4.1.2** bool memeEspeceEnVue()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de savoir si un agent de la même espèce se trouve dans le champ de vision de l’agent | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’un agent de la même espèce soit en vue | Lancer la simulation et ajouter un agent. Si un agent de la même espèce est en vue, la fonction retourne Vrai |

* **Id4.1.3** Agent getAgentMemeEspece()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de retourner un agent de la même espèce qui est dans le champ de vision de l’agent. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini et qu’un agent de la même espèce soit en vue. | Lancer la simulation et ajouter un agent. Si un agent de la même espèce est en vue, retourne cet agent. |

* **Id4.1.4** void Mimetisme()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’effectuer le mimétisme. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si l’agent est bien défini, qu’un agent de la même espèce soit en vue et que l’action de l’agent en visuel vérifie les conditions nécessaire à la copie | Lancer la simulation et ajouter un agent. Si un agent de la même espèce est en vue, l’observateur imite l’action observée s’il augmente ses chances de survie. |

## 5) Téléportation au-dessus d’un agent

1. Fonction de téléportation

* **Id5.1.1** void Onclick()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet qui appelle une fonction “TeleportNomAgent”qui nous permet de nous déplacer au-dessus d’un agent de l’espèce dont on a choisi. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier si le panel de téléportation est bien défini et qu’il existe un agent de l’espèce choisie . | Lancer la simulation et ajouter un agent. Appuyer sur un des boutons dans le panel de téléportation, si un agent de cette espèce existe, la fonction nous déplace au-dessus d’un agent de cette espèce, sinon ne fait rien. |

* **Id5.1.2** void AjouterAgentsList()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’ajouter tous les agents dans une liste commune et dans liste propre à leur espèce. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier que les agents soient bien définis et qu’il existe des agents | Lancer la simulation et ajouter un agent dans la simulation. L’agent sera ajouté dans la liste commune et à la liste propre à son espèce. |

* **Id5.1.3** void toggleTeleportPanel()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’activer ou désactiver le panel de téléportation. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier que le panel de téléportation existe bien et que les autres panels soient bien définis. | Lancer la simulation et appuyer sur les boutons du panel de téléportation. |

* **Id5.1.4** void TeleportAgent()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui nous place au-dessus d’un agent qui se trouve dans la liste commune des agents. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier que les agents soient bien définis et qu’il existe des agents autres que les autotrophes dans la simulation. | Lancer la simulation et appuyer sur la touche T pour se déplacer sur un agent qui se trouve dans la liste |

* **Id5.1.5** void TeleportAgent()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui nous place au-dessus d’un agent qui se trouve dans la liste commune des agents. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier que les agents soient bien définis et qu’il existe des agents autres que les autotrophes dans la simulation. | Lancer la simulation et appuyer sur la touche T pour se déplacer sur un agent qui se trouve dans la liste. |

* **Id5.1.6** void Teleport”NomAgent”()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui nous place au-dessus d’un agent qui se trouve dans la liste d'espèces de l’agent choisi. | Dépendance de la classe Agent pour ce test unitaire. | Vérifier que les agents soient bien définis et qu’il existe des agents de l’espèce choisie dans la simulation | Lancer la simulation et appuyer sur une image dans la panel de téléportation pour se déplacer sur un agent de l’espèce. |

* **Id5.1.7** void RemoveFromList()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de retirer les agents qui ont été supprimés de la simulation | Dépendance de la classe Agent pour ce test unitaire. | Vérifier que les agents soient bien définis et qu’un agent soit supprimé de la simulation. | Lancer la simulation et supprimer un agent avec la fonction deleteAgent() ou attendre qu’un agent meure et soit détruit. |

## 6)Suppression d’agent

1. Fonction de suppression d’agent

* **Id6.1.1** void deleteAgent()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de supprimer un agent de la simulation | Dépendance de la classe Agent pour ce test unitaire. | Vérifier qu’un agent existe dans la simulation. | Lancer la simulation, cliquer sur un agent, appuyer sur le bouton + d’AgentStatPanel puis cliquer sur le bouton “Supprimer Agent”. |

## 7)Système de peur

1. Action de peur

* **Id7.1.1** void run()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet à l’agent de s’enfuir dans la direction opposée d’un prédateur à cause de la peur. | Dépendance de la classe Agent pour ce test unitaire. | L’agent effectuant cette action doit avoir une peur supérieure à 10% et un prédateur doit être en vue. | Lancer la simulation, augmenter la peur de l’agent à 10% ou plus et placer un prédateur dans son champ de vision. |

* **Id7.1.2** void terrified()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui immobilise un agent à cause de la peur | Dépendance de la classe Agent pour ce test unitaire. | L'agent qui effectuant cette action doit avoir une peur supérieure à 99%. | Lancer la simulation, augmenter la peur de l’agent à 99% ou plus. |

## 8) Sauvegarde et chargement d’une simulation passé

1. Sauvegarde d’une simulation

* **Id8.1.1** SaveAgentsToJson()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de sauvegarder les données de chaque agent dans des fichiers | Dépendance de la classe Agent pour ce test unitaire. | Vérifier que des agents soient bien définis. | Lancer la simulation, ajouter des agents dans la simulation, ouvrir le menu de pause et cliquer sur “Enregistrer la simulation” |

1. Chargement d’une simulation

* **Id8.1.1** OnClickAddAgents()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de charger les agents dont les données ont été enregistrées dans des fichiers | Dépendance de la classe Agent pour ce test unitaire. | Vérifier qu’il existe des fichiers dans le dossier SaveFolder | Lancer une ancienne simulation et vérifier que les agents ont bien été créés. |

## 9)Fonction concernant l’interface utilisateur

1. Activation et désactivation d’objet dans la simulation

* **Id9.1.1** OpenUI()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet d’activer un objet dans la simulation | Aucune dépendance de classe | Vérifier que le “gameobject” soit défini | Lancer la simulation appuyer sur un bouton ayant le script. |

* **Id9.1.2** CloseUI()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de désactiver un objet dans la simulation | Aucune dépendance de classe | Vérifier que le “gameobject” soit défini. | Lancer la simulation appuyer sur un bouton ayant le script. |

1. Déplacement d’un panel

* **Id9.2.1**  OnDrag()

| Description | Contraintes | Dépendances | Procédure de test |
| --- | --- | --- | --- |
| Fonction qui permet de rendre un panel déplaçable | Aucune dépendance de classe | Vérifier que le “gameobject” soit défini. | Lancer la simulation et déplacer un panel ayant le script. |

# 

# IV. Bibliographie

*Contexte et Historique*

Équilibre proies-prédateurs:

<https://fr.wikipedia.org/wiki/%C3%89quilibres_pr%C3%A9dateurs-proies>

Équations de prédation de Lotka-Volterra:

<https://fr.wikipedia.org/wiki/%C3%89quations_de_pr%C3%A9dation_de_Lotka-Volterra>