THE SILENT HUNT 2025

Par Hugo Rochedreux, Noah Laghlali et Emma Klingler
Tuteur : Monsieur Olivier Perrin
07/11/2025

Trello : https://trello.com/b/b9XkBREh/the-silent-hunt
Github :

Présentation du projet

The Silent Hunt est un jeu de survie développé sur Roblox Studio en langage Lua.
Le joueur incarne un lapin, seul dans une forêt silencieuse où rôde un chasseur contrôlé par une intelligence artificielle.
L'objectif : survivre le plus longtemps possible en restant discret, en gérant sa faim, son stress et ses ressources.

Le cœur du jeu repose sur une IA évolutive capable d'apprendre des habitudes du joueur pour mieux les anticiper.
Au fil des parties, le chasseur devient plus rusé, rendant chaque session différente et imprévisible.

Le jeu propose à la fois un mode solo, pour une expérience immersive et tendue, et un mode multijoueur, axé sur la coopération et la stratégie entre survivants

Etude du produit

https://www.canva.com/design/DAG3vundYXk/Oo9d0JkTnNmkeKKEJ0_sEQ/edit?utm_content=DAG3vundYXk&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Résumé du projet

The Silent Hunt est un jeu de survie asymétrique développé sur la plateforme Roblox. Le joueur y incarne un lapin poursuivi par un chasseur contrôlé par une intelligence artificielle évolutive.
L'objectif est de survivre dans un environnement hostile, où chaque son, trace ou mouvement peut trahir la présence du joueur. L'intelligence artificielle s'adapte progressivement aux comportements du joueur, rendant chaque partie unique.
Le jeu propose une expérience immersive et stratégique, alliant tension psychologique, environnement vivant et progression narrative.

Maquette


Cobap - Je vais être le dernier en vie vous aller voir ;)
MrPichou - Tu vas me servir de leurre c'est tout ce qu'il va se passer
Appuie sur / pour parler dans le chat

Description d’une partie type :

Le jeu se lance on commence dans un menu, on peut choisir entre un menu solo ou multijoueur, ensuite on peut choisir une difficulté entre facile, moyen et difficile.
Facile : Le chasseur se déplace aléatoirement et chasse un lapin si il est proche
Moyen : BT le chasseur est un peu plus complexe avec un arbre de décision
Difficile : Machine learning, AI plus évolué qui apprend

Lancement partie :
Les lapins démarrent aléatoirement dans la forêt, le joueur est un des lapins.
Le chasseur démarre au centre de la map dans sa cabane, il prend un petit peu de temps avant de commencer à chasser, se prépare pendant 30s environ.
Le but des lapins est de survivre un cycle jour nuit soit environ 10-20 min (pas encore défini), les lapins ont une barre de vie, une barre faim et perdent de la faim avec le temps, ils doivent donc se nourrir à l’aide de baie ou autre nourriture trouvée dans la forêt.
Les lapins peuvent se déplacer en sautant et sont plus rapides que le chasseur, ils peuvent rentrer dans un terrier (une galerie souterraine avec plusieurs entrées et sorties).
Le chasseurs lui peut patrouiller, entendre les bruits et essayer de prédire les actions des lapins, il peut tirer si un lapin est assez proche avec un temps de recharge assez long 5-10s.
Il possède un nombre de pièges (3 par exemple) il peut les placer où il veut et si un lapin tombe dedans, le chasseur sera informé et le lapin est bloqué tant qu’un autre lapin ne l’aide pas (les lapins peuvent s’aider).
La partie continue tant qu’il reste un lapin en vie et que le temps n’est pas fini.

Fonctionnalités

1 . Fonctionnalités primaires

1. Contrôle et déplacement du lapin

- Le joueur contrôle un lapin en vue subjective : FPS.
- Déplacements libres avec gestion du sprint, du bruit et de la fatigue.
- Interaction avec l'environnement (manger, creuser, se cacher).
- Physique simplifiée mais cohérente : gravité, collisions, sauts, glissades légères sur terrain humide.

2. Gestion des systèmes vitaux

Trois jauges principales influencent la survie :

- Faim : baisse constante avec le temps, régénérable en mangeant.
- Vie : affectée par la faim extrême, les pièges, ou la capture par le chasseur.
- Stress : augmente à proximité du chasseur ou lors d'événements dangereux ; diminue dans les zones sûres ou avec des actions calmantes.

→ Ces jauges influencent directement la vitesse, le bruit et la perception du joueur.

3. Système d'intelligence artificielle du chasseur

- Architecture en FSM (Finite State Machine) pour gérer les comportements : patrouille, poursuite, embuscade, repos.
- Évolution vers un arbre de décision pour la gestion contextuelle (vision, bruit, mémoire, zones fréquentées).
- Progression vers une IA adaptative qui apprend des schémas comportementaux des joueurs (zones visitées, tactiques répétées).
- Gestion des stimuli : vision, sons, traces de pas, odeurs.

→ L'IA analyse les actions du joueur pour adapter sa stratégie, plaçant le chasseur comme un adversaire imprévisible.

4. Pathfinding et navigation intelligente

- Utilisation du PathFindingService de Roblox pour calculer les déplacements du chasseur.
- Prise en compte de la topographie, des obstacles et des zones interdites.
- Détection dynamique des changements de terrain (effondrement, pièges déclenchés, etc.).

5. Système de détection sonore et de traces

- Chaque action du joueur génère un niveau sonore proportionnel à la vitesse et au type de surface.
- L'IA réagit en fonction de l'intensité et de la distance du son.
- Les traces de pas ou d'interactions (herbe aplatie, nourriture manquante) servent d'indices persistants pour la traque.

6. Mécanique de cachettes et terriers

- Terriers naturels ou creusés par le joueur.
- Réduction du stress et possibilité de "disparaître" temporairement du champ de perception du chasseur.
- Risque d'embuscade si le terrier est trop utilisé ou détecté par l'IA.

7. Interaction sonore et objets environnementaux

- Le joueur peut créer des leurres pour détourner l'attention de l'IA (bruit, odeur, mouvement).
- Le chasseur peut lui-même poser des pièges sonores ou visuels.
- Tous ces éléments alimentent une chaîne d'événements réactive entre joueur et IA.

2. Fonctionnalités secondaires

1. Mode multijoueur

- Synchronisation client–serveur via RemoteEvents et RemoteFunctions
- Gestion collective des jauges (stress de groupe, zones partagées).

- Coopération : sauvetage, distraction, partage de ressources.
- IA adaptative multi-cibles : le chasseur apprend aussi des comportements collectifs.

2. Objets interactifs et ressources naturelles

- Nourriture : plantes, baies, champignons, carottes sauvages.
- Objets à fabriquer ou à collecter : leurres, filets, balises.
- Ressources limitées pour encourager la prise de risque et l'exploration.

3. Cycle jour/nuit et météo dynamique

- Effets météo (pluie, vent, brouillard) influençant les sons et la visibilité.

4. Missions secondaires et progression

- Objectifs ponctuels : explorer, récupérer un artefact, sauver un autre lapin.
- Système de récompenses : réduction de stress, amélioration de capacités, buff temporaire.
- Progression symbolique (niveaux ou jours de survie).

5. Système de sauvegarde et statistiques

- Sauvegarde automatique avec DataStoreService
- Statistiques : jours survécus, pièges évités, zones explorées.
- Suivi de la progression du chasseur (niveau d'adaptation etc).

6. Simulation IA Lapin (entraînement du chasseur)

- Génération de lapins virtuels simulant différents styles de jeu.
- Données de comportement enregistrées et réutilisées pour entraîner le chasseur.
- Permet de "faire évoluer" le chasseur même hors session réelle.

Recensement et évaluation des risques

- Intégration et synchronisation de l'IA évolutive (client vs serveur) | Impact : Élevé |Priorité : Haute
  La logique d'IA peut se retrouver mal répartie entre clients et serveur, provoquant des comportements incohérents, des bugs de synchronisation ou des désaccords d'état entre joueurs.

- IA perçue comme trop forte, omnisciente ou frustrante | Impact : Élevé | Priorité : Haute
  si l'IA semble tricher (connaître la position exacte du joueur ou punir sans délai), les joueurs ressentiront une injustice et quitteront la partie.

- Entraînement ML coûteux et non reproductible (100/10 000 runs) | Impact : Moyen | Priorité : Moyenne
  les sessions d'entraînement demandées pour améliorer l'IA peuvent consommer beaucoup de temps et de ressources, et produire des résultats difficiles à reproduire sans pipeline bien défini.

- Mécaniques (stress, terriers, faim) mal équilibrées → frustration ou jeu trop facile | Impact : Élevé | Priorité : Haute
  des valeurs mal calibrées rendent le jeu soit trop punitif (frustrant) soit trop laxiste (sans enjeu), détruisant l'expérience prévue.

- Complexité cognitive trop haute (joueurs perdus par des mécanismes multiples) | Impact : Moyen | Priorité : Moyenne
  l'ensemble des systèmes risque de submerger le joueur novice, rendant l'accès au gameplay difficile et détériorant la courbe d'apprentissage.

- Triche / exploitation (clients manipulant position, inventaire, invisibilité) | Impact : Élevé | Priorité : Haute
  le risque que des joueurs modifient le client pour tricher.

- Sous-estimation du temps pour IA adaptative / ML (dépassement planning) | Impact : Élevé

- Priorité : Haute
  le développement de l'IA adaptative peut prendre bien plus de temps que prévu,

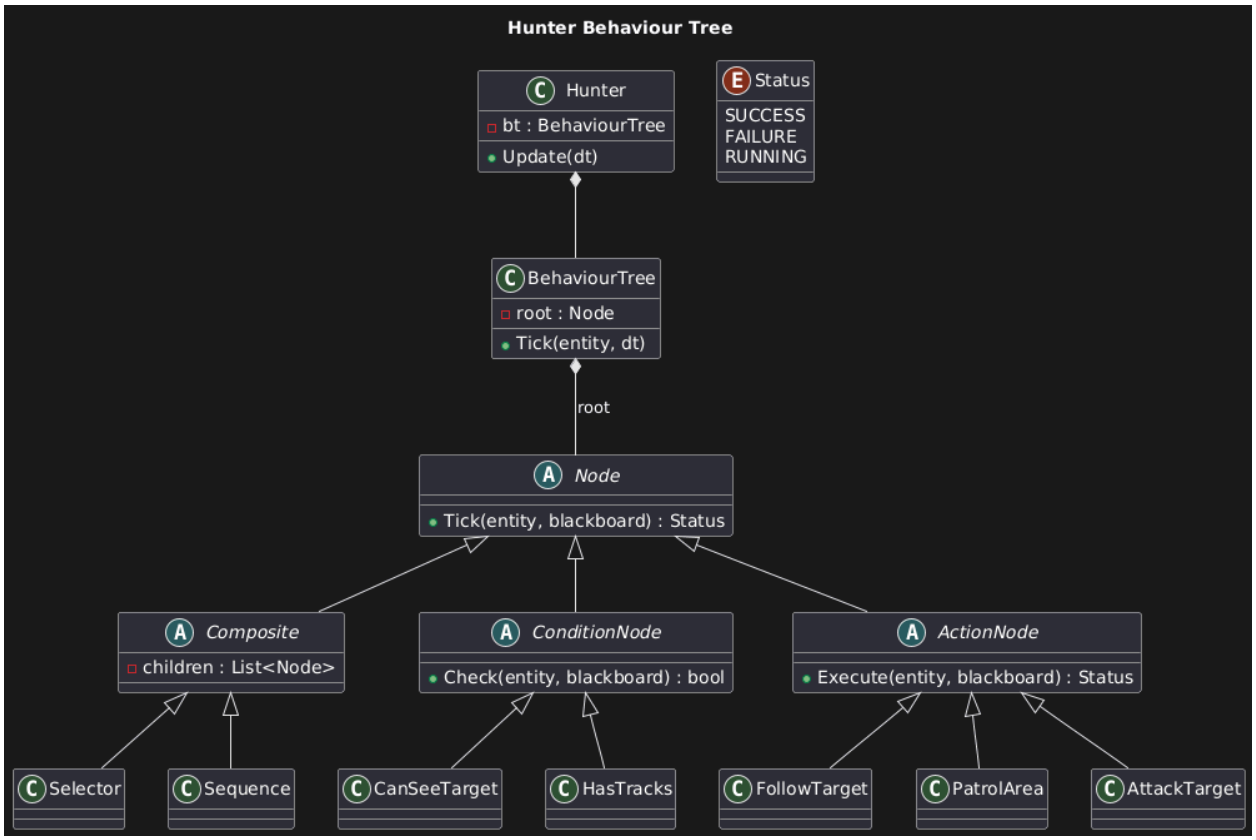
entraînant retard de versions et renoncements à d’autres fonctionnalités.

- Faible rétention si gameplay répétitif | Impact : Élevé | Priorité : Haute
  si les parties deviennent redondantes (mêmes tactiques, mêmes cartes), les joueurs ne reviendront pas, affectant la longévité et le succès du jeu.

Abre de décision

Roblox fournit le pathfinding (comme Unity), mais ce que nous développons, c'est la logique qui utilise ce système : gestion de la perception, décision comportementale, transitions de state, recalcul dynamique, prise en compte des stimuli, adaptation du comportement, apprentissage… Nous enveloppons PathfindingService dans une classe Navigator qui gère les déplacements, les recalculs et les interactions avec le FSM.

Le FSM est composé d'une StateMachine qui délègue le comportement au currentState via la méthode Update(). Chaque état implémente Update() et peut retourner le nom d'un état cible. La StateMachine réalise ensuite la transition. Ainsi, le comportement reste encapsulé dans les classes d'état, permettant une IA modulaire, lisible et extensible. Pathfinding, perception, navigation et décision restent orchestrés par l'IA via des services dédiés.



Rééquilibrage de la jouabilité

1. IA du chasseur

Le chasseur conserve son rôle central mais il n'est plus une machine imbattable.

1. Vision et perception

Le chasseur possède un champ de vision de 70°. Sa portée visuelle maximale est de 15 à 20 mètres, mais elle chute à 10 mètres en cas de brouillard ou de crépuscule.Les zones d'ombre et les buissons bloquent totalement sa vision, sauf si le lapin produit un bruit supérieur à un certain seuil.Avant chaque action (tir, poursuite, investigation), l'IA a une latence de 1 à 1,5 seconde,ce qui rend son comportement plus réaliste et moins fort.

2. Audition

Le chasseur entend les sons dans un rayon d'environ 25 mètres.Son audition n'est pas parfaite : il peut se tromper de direction d'une trentaine de degrés, et lorsqu'il entend plusieurs sons rapprochés, il choisit parfois la mauvaise source.
L'IA ne réagit qu'à des bruits supérieurs à un certain nombre de décibels (30 ?) et met une à deux secondes à se diriger vers la zone suspecte.

3. Gadgets et outils

L'utilisation d'outils a été fortement restreinte pour éviter les abus.Le scanner thermique ne peut être activé que deux fois par jour et ne révèle les lapins visibles que pendant trois secondes.Les mines sonores et autres pièges ont des temps de recharge plus longs et une visibilité moindre, permettant au joueur de les éviter ou de les désactiver.Le brouilleur de zone(pour que les lapins ne communiquent pas) ne peut être utilisé qu'une fois par minute et n'a qu'une portée de cinq mètres.L'IA possède es caméras animales, mais ne peut en surveiller que deux à la fois.

4. Mémoire et apprentissage

La mémoire du chasseur est désormais limitée. Il conserve en mémoire les trois dernières zones où il a détecté une activité suspecte, et oublie les traces anciennes au bout de 90 secondes.Il ne pose plus ses pièges directement sur un terrier, mais à une distance de sécurité de 5 à 10 mètres, ce qui laisse au joueur la possibilité d'en sortir sans mort instantanée.

2. Lapin

Le lapin devient un acteur stratégique,et pas une simple proie.

1. Déplacements et actions de base

Le joueur peut marcher, ramper ou courir, chaque mode influençant le bruit et la fatigue.
Marcher ou ramper génère très peu de son et reste quasi indétectable, tandis que courir crée un fort bruit, augmente la faim et le stress, mais permet de s'échapper plus vite.
Le lapin peut sauter pour éviter les pièges au sol, creuser un terrier en dix secondes pour créer un abri, manger pour réduire sa faim et son stress, ou observer les alentours pour détecter les pièges.

2. Camouflage et cachettes

Le lapin peut se cacher dans des buissons pour devenir totalement invisible à condition d'y entrer lentement. Une entrée trop rapide provoque un bruit capable d'alerter le chasseur.
Les terriers ne servent plus seulement de cachettes, ils deviennent des téléporteurs.
Chaque utilisation consomme environ 15 % de la jauge de faim et ne peut être répétée qu'après un certain délai, pour éviter l'abus.Aussi, ils sont limités en temps d'utilisation mais peuvent contenir certains bonus..Certains terriers débouchent dans des zones risquées, ce qui maintient la tension et le hasard stratégique.

3. Fabrication de pièges

Le lapin peut fabriquer ses pièges directement depuis l'environnement, **avec un inventaire** Le système repose sur une interaction contextuelle : lorsqu'il se trouve près d'éléments exploitables (branches, pierres, racines, lianes…), une icône d'action s'affiche ("Maintenir E pour fabriquer").

Branches : tombées au sol, elles servent à construire des collets ou à créer des leurres sonores.**Lianes** : suspendues aux arbres, elles sont utilisées pour tisser les collets végétaux.**Pierres** : trouvées près des rivières, des falaises ou dans les ruines, elles servent à stabiliser les pièges à tronc ou déclencher des chocs sonores.**Feuilles ou touffes d'herbe sèche** : récupérées lors de la fouille du sol, elles permettent de fabriquer des leurres visuels ou de camoufler un piège pour le rendre indétectable.

Chaque ressource ramassée occupe une petite place dans l'inventaire du lapin (4 objets maximum ?).

→ **Construction :**

Le joueur maintient la touche d'action pendant quelques secondes pour assembler le piège :

- Collet végétal : le lapin tresse une liane autour d'une branche flexible :ralentit le chasseur pendant 3 secondes.

- Piège à tronc (ou branches) : le lapin pousse un tronc mort pour barrer un passage ou piéger un couloir: bloque physiquement un chemin

- Piège sonore : le lapin place une pierre suspendue ou une branche prête à tomber, provoquant un bruit s'il la déclenche à distance. : L'IA part vérifier la zone

Chaque fabrication fait un petit bruit localisé, forçant le joueur à choisir un moment calme pour agir. Aussi, fabriquer un piège consomme de la faim et augmente les autres jauges de vie.

4. Statuts et variables du joueur

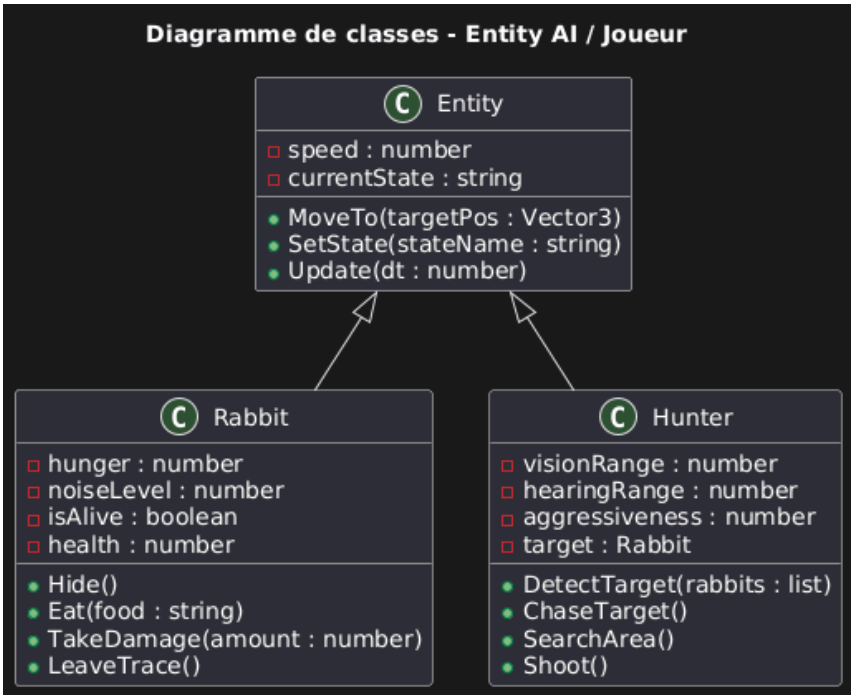
Le stress, la faim et la vie forment les trois piliers de la survie. La faim diminue naturellement, et une fois vide, elle fait baisser la vie et la vitesse.Le stress augmente à proximité du chasseur ou lors d'actions bruyantes mais à un niveau critique, il altère la vision et provoque des crises de panique.

5. Environnement et interactions

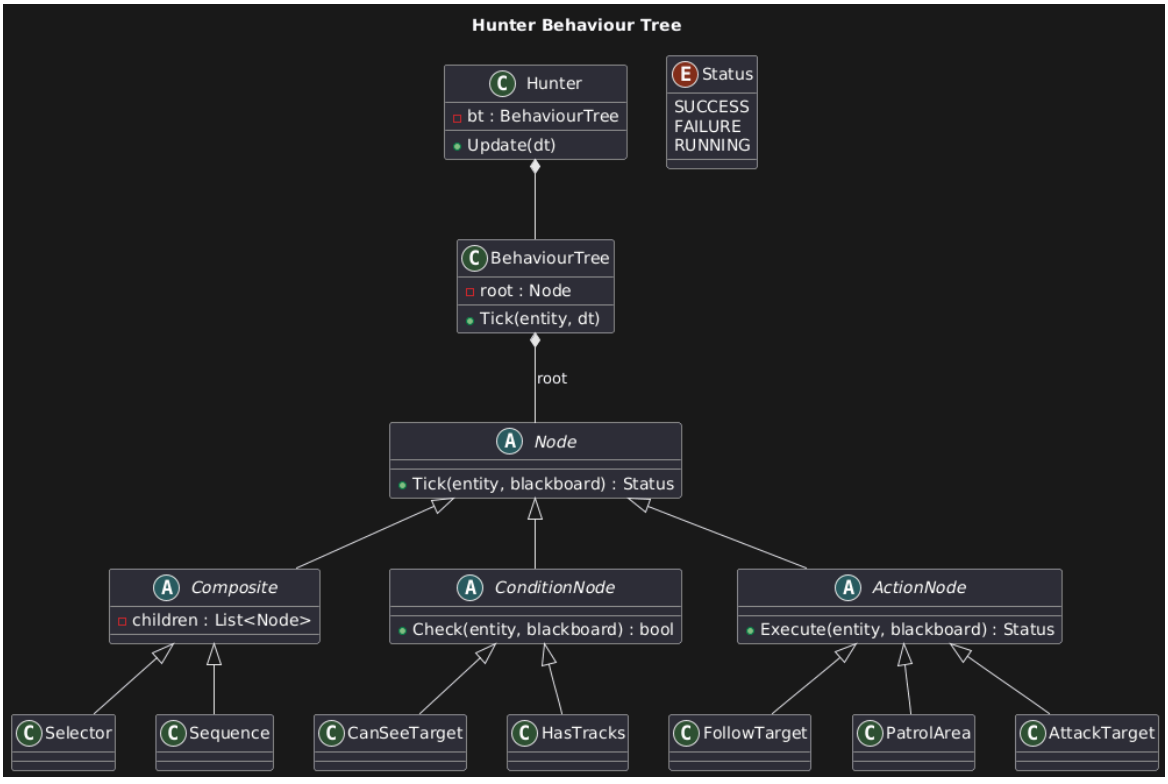
L'environnement est un système où chaque zone influence directement la survie. Certains éléments naturels peuvent aussi accorder des **bonus temporaires** : des plantes énergétiques augmentent la vitesse ou réduisent le stress, des herbes médicinales restaurent partiellement la vie,et des fruits rares permettent de régénérer la faim plus efficacement. Ces ressources sont souvent situées dans des zones exposées,ou bien visibles après l'accomplissement d'une mission, obligeant le joueur à choisir entre le risque et la récompense.

Diagrammes de classe

En premier nous avons fait un diagramme de classe pour les entity joueur et chasseur pour expliquer leur attributs et méthodes



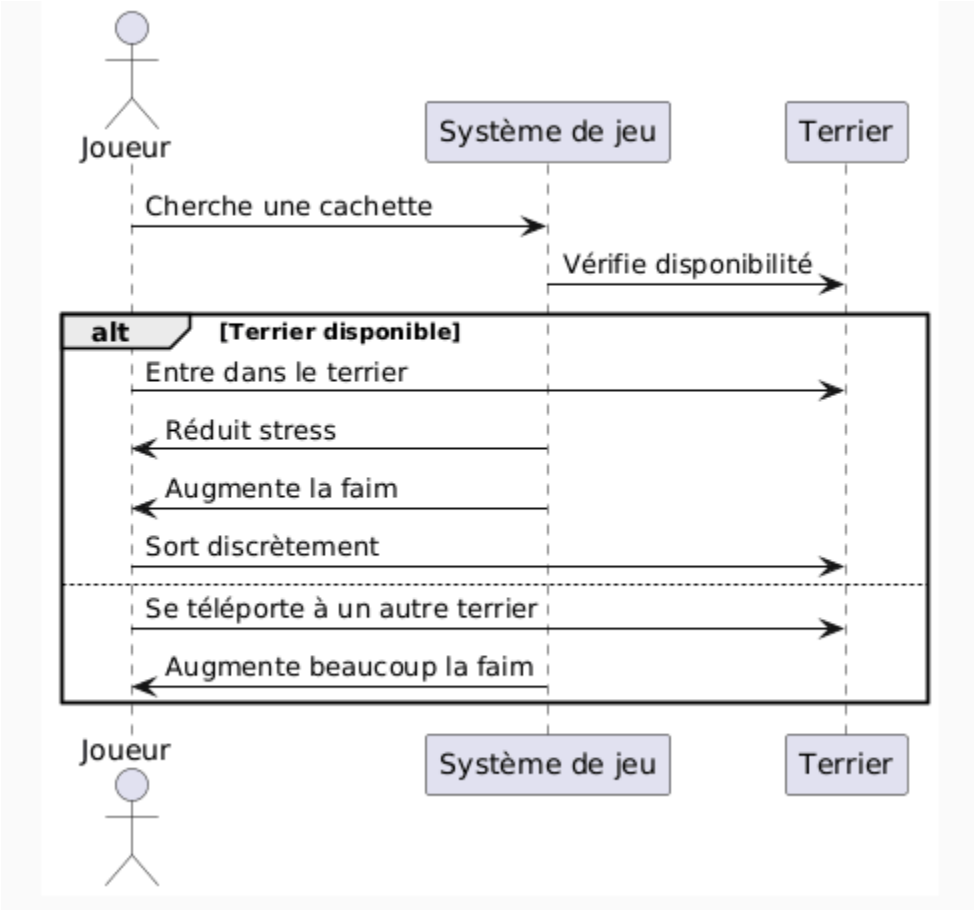
Nous avons aussi un diagramme de classe pour le hunter par exemple, donc nous avons des Noeuds de type composite, condition ou action. Ce diagramme de classe nous sera très utile des les premières itérations aussi bien côté hunter que côté lapin AI pour le machine learning



On a fait des diagrammes de séquence pour montrer l'ordre des échanges entre les objets et le serveur. Ils permettent de comprendre comment les actions se déroulent dans le temps et comment les différents éléments du jeu communiquent entre eux.

Diagramme lapin utilise un terrier :

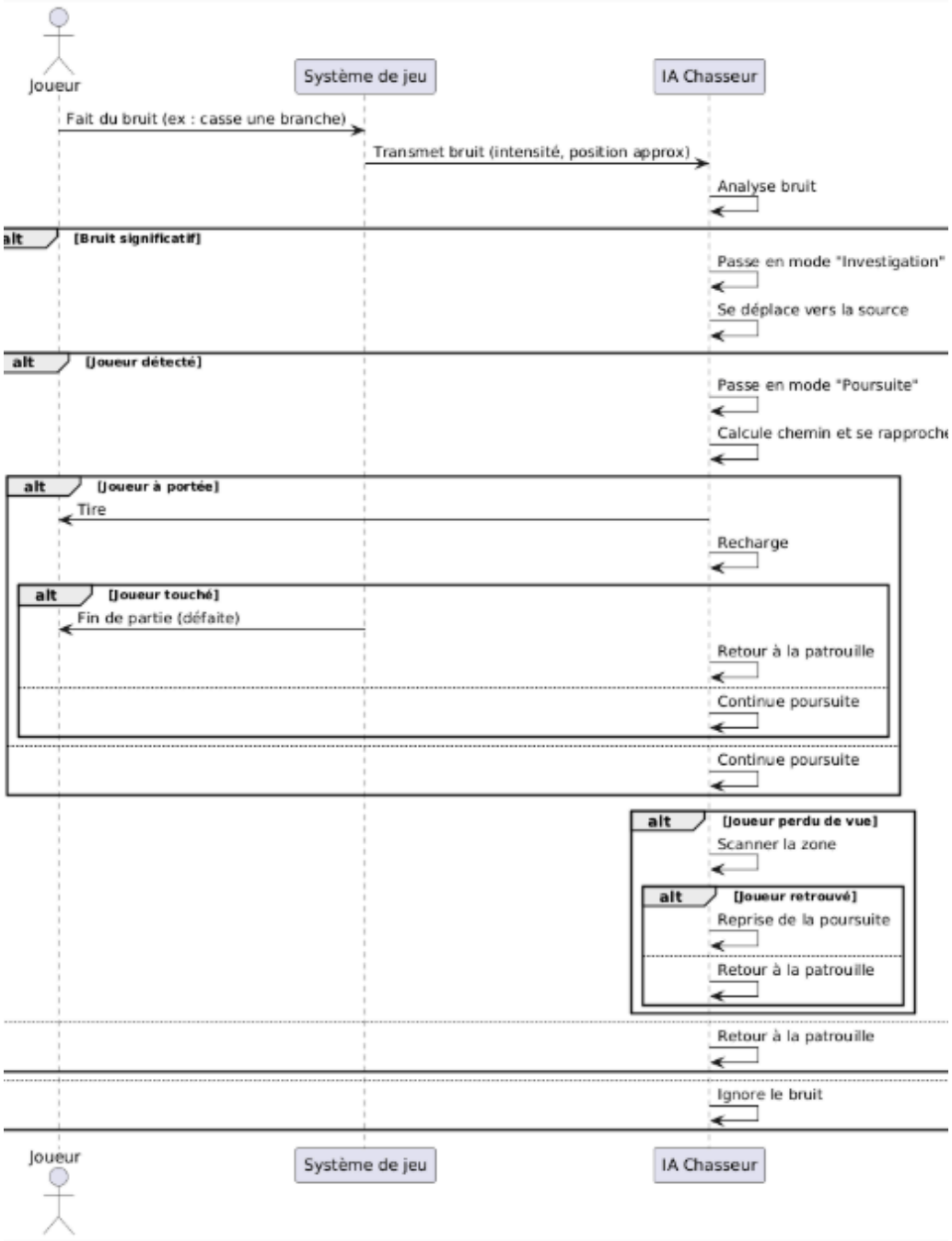
Le principe de terrier peut être un peu complexe, donc ce diagramme est important pour pouvoir bien comprendre son fonctionnement.



Lorsque le joueur cherche une cachette, le système de jeu interroge un terrier pour vérifier s'il est disponible. Si le terrier est libre, le joueur peut s'y réfugier. Cette action réduit son niveau de stress, mais augmente légèrement sa faim progressivement (plus qu'en temps normal). Une fois calmé, le joueur ressort discrètement du terrier pour ne pas mourir de faim.

En revanche, si le joueur le souhaite, il peut se téléporter vers un autre terrier au lieu de simplement sortir. Cette téléportation, plus brutale et énergivore, entraîne une augmentation beaucoup plus importante de sa faim.

Diagramme comportement du chasseur :



Lorsqu'un joueur fait du bruit (par exemple en cassant une branche), le système de jeu capte cette action et transmet les informations au chasseur, notamment l'intensité du bruit et une position approximative.

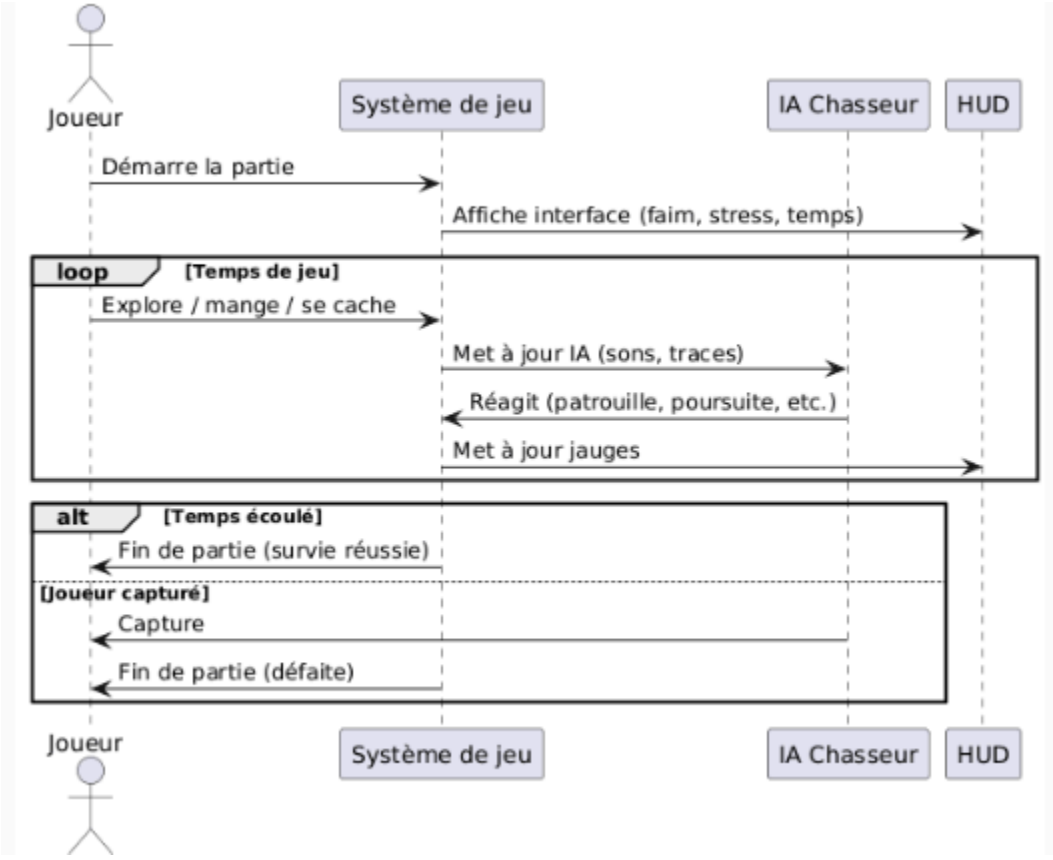
L’IA analyse alors ce bruit. Si il est jugé significatif, le chasseur passe en mode investigation et se dirige vers la source du bruit. Une fois sur place, il tente de détecter visuellement le joueur.

Si le joueur est détecté, le chasseur entre en mode poursuite. Il calcule un chemin pour se rapprocher de sa cible. Si le joueur est à portée, le chasseur ouvre le feu, puis recharge son arme.

Si le tir touche le joueur, la partie se termine par une défaite pour le joueur, et le chasseur retourne à sa patrouille. Si le tir échoue, la poursuite continue.

Dans le cas où le joueur parvient à échapper à la vue du chasseur, celui-ci scanne la zone pour tenter de le retrouver. Si le joueur est repéré à nouveau, la poursuite reprend. Sinon, le chasseur abandonne et retourne à sa patrouille.

Diagramme déroulement d’une partie



La partie commence lorsque le joueur initie le jeu. Le système de jeu affiche alors l’interface utilisateur via le HUD, présentant les jauges essentielles telles que la faim et la vie, mais aussi le temps et le stress via des affichages plus subtiles.

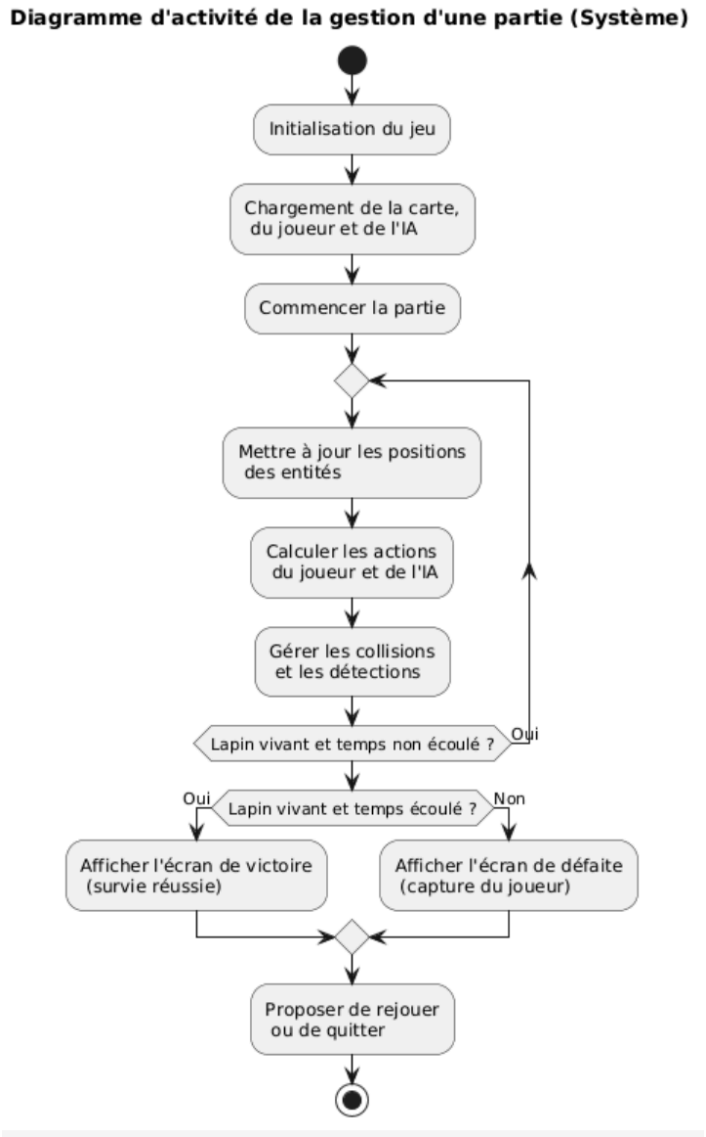
Pendant toute la durée de la partie, une boucle s’exécute : le joueur explore l’environnement, recherche de la nourriture ou tente de se cacher. À chaque action, le système met à jour le chasseur en lui transmettant les informations pertinentes comme les sons ou les traces laissées par le joueur.

L’IA réagit en conséquence, adaptant son comportement entre patrouille, poursuite ou investigation selon les informations reçues. En parallèle, le système actualise les jauges du HUD pour refléter l’état du joueur en temps réel.

Lorsque le temps imparti s’écoule, la partie se termine par une victoire du joueur, symbolisant une survie réussie. En revanche, si le chasseur parvient à capturer le joueur avant la fin du temps, la partie se conclut par une défaite

Diagrammes d'activité

On a fait des diagrammes d’activités pour montrer le déroulement des actions dans le jeu, étape par étape. Ils permettent de visualiser la logique et les décisions du lapin, du chasseur et du système avant de les coder.

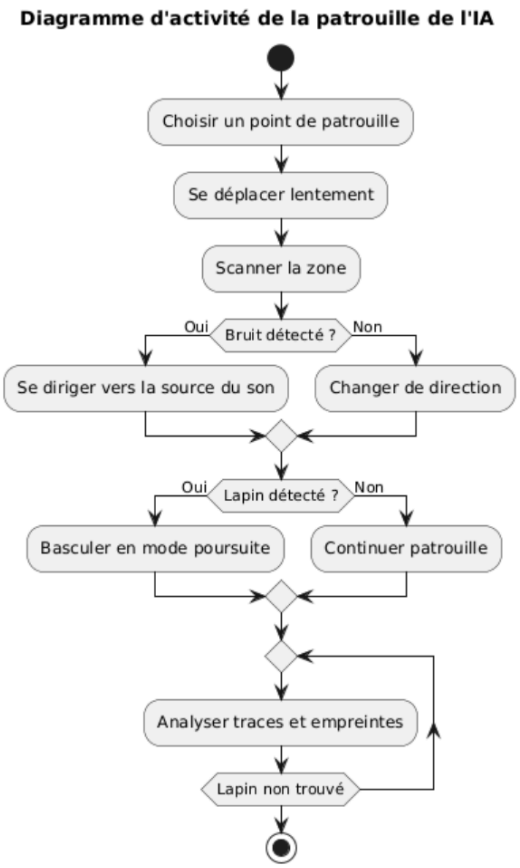


Le jeu commence par l’initialisation du système.
La carte, le joueur (le lapin) et l’IA du chasseur sont chargés. Une fois tout prêt, la partie débute.

Pendant la partie, le système met à jour en continu les positions du joueur et de l’IA.
Il calcule leurs actions respectives, comme se déplacer, poursuivre, ou se cacher, et gère les collisions entre eux ou avec les éléments du décor.

Cette boucle se répète tant que le lapin est vivant et que le temps n’est pas écoulé.
À chaque instant, le système suit le temps de jeu et surveille l’état du joueur.

Si le lapin réussit à survivre jusqu’à la fin du temps, un écran de victoire s’affiche.
S’il se fait capturer par le chasseur, c’est l’écran de défaite qui apparaît.À la fin de la partie, le système propose au joueur de recommencer une nouvelle session ou de quitter le jeu.

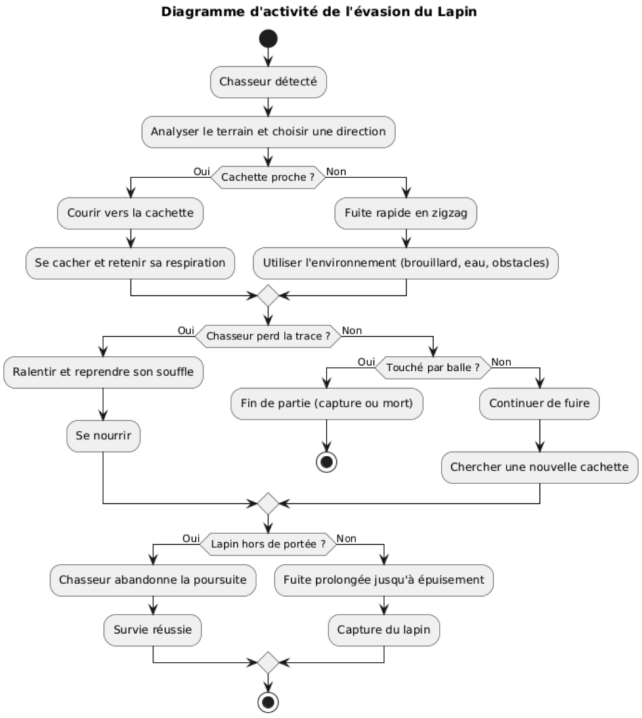


Le chasseur commence sa patrouille en choisissant un point à atteindre. Elle se déplace lentement vers ce point tout en scannant régulièrement la zone pour détecter toute activité suspecte.

Si un bruit est détecté, l'IA interrompt sa trajectoire et se dirige vers la source du son pour en identifier l'origine. Si aucun bruit n'est perçu, elle change simplement de direction et poursuit sa patrouille.

Une fois sur place, elle vérifie si le lapin est visible. Si le lapin est repéré, l'IA bascule immédiatement en mode poursuite, abandonnant sa routine de patrouille pour tenter de capturer sa cible.

Si le lapin n'est pas détecté, l'IA continue sa patrouille tout en analysant les traces et empreintes laissées dans l'environnement. Si ces indices ne permettent pas de localiser le lapin, elle conclut que celui-ci n'a pas été trouvé et reprend son cycle de surveillance.

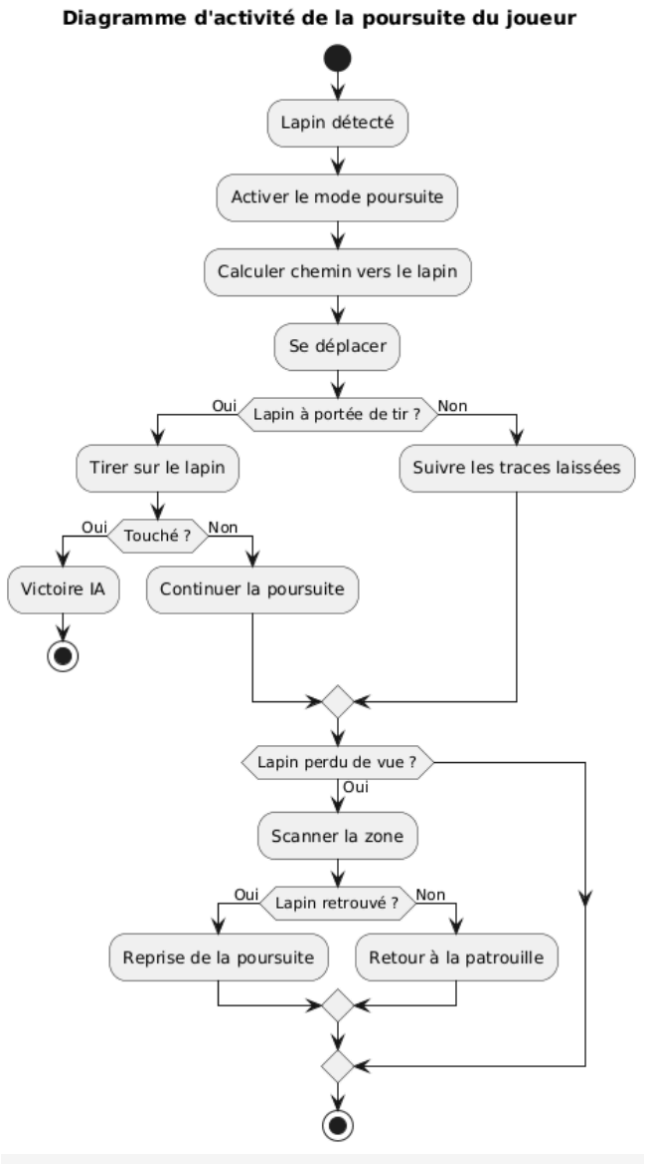


Lorsque le lapin détecte la présence du chasseur, il analyse rapidement le terrain pour choisir une direction de fuite. S'il repère une cachette proche, il s'y dirige en courant, pour essayer de semer son poursuivant. Une fois à l'abri, il se cache.

Si aucune cachette immédiate n'est disponible, le lapin utilise les éléments de l'environnement (comme le brouillard, l'eau ou des obstacles) pour brouiller les pistes et ralentir le chasseur.

Si le chasseur perd sa trace, le lapin peut ralentir, reprendre, se nourrir.... En revanche, si le chasseur le poursuit toujours, plusieurs issues sont possibles : le lapin peut être touché par une balle, ce qui entraîne sa capture ou sa mort, ou bien il continue de fuir en cherchant une nouvelle cachette.

Si le lapin parvient à sortir de la portée du chasseur, l'évasion est un succès.



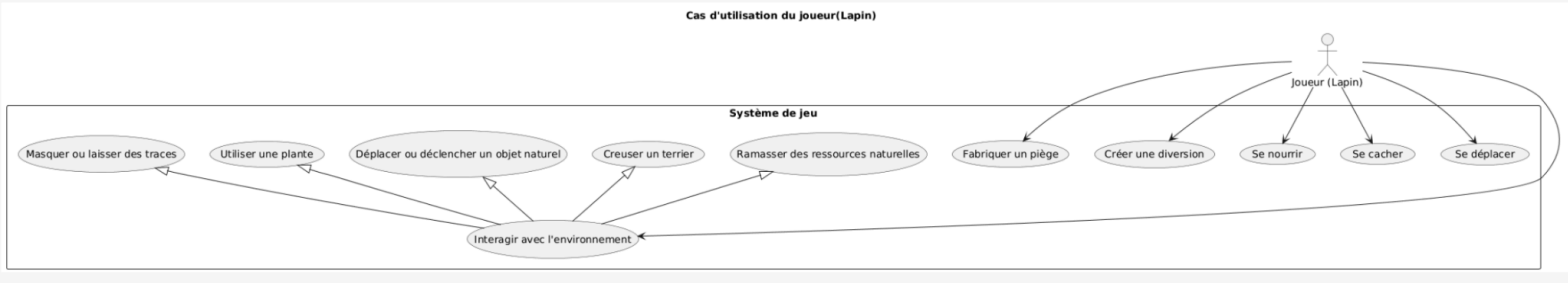
Lorsque le lapin est détecté, l'intelligence artificielle active immédiatement le mode poursuite. Elle commence par calculer le chemin optimal pour atteindre sa cible, puis se met en mouvement pour réduire la distance.

Si le lapin reste visible, l'IA tente de le toucher en tirant. Si le tir est réussi, la partie se termine par une victoire de l'IA (en mode solo). Si le tir échoue, la poursuite continue.

Dans le cas où le lapin disparaît du champ de vision, l'IA lance un scan de la zone pour tenter de le retrouver. Si elle y parvient, la poursuite reprend. Sinon, elle abandonne et retourne à sa routine de patrouille.

Diagrammes de Cas d’utilisation

On a fait des diagrammes de cas d’utilisation pour montrer les interactions entre les acteurs et le jeu. Ils servent à définir clairement ce que chaque entité peut faire avant de passer à la conception technique.



Se déplacer = Le joueur peut marcher, courir, ramper ou sauter selon la situation ; chaque mode de déplacement génère un niveau de bruit différent et influence la détection par le chasseur.

Se cacher = Le lapin peut se dissimuler dans des buissons, troncs creux ou terriers pour réduire sa visibilité et son empreinte sonore, mais ces cachettes ne garantissent jamais une sécurité totale.

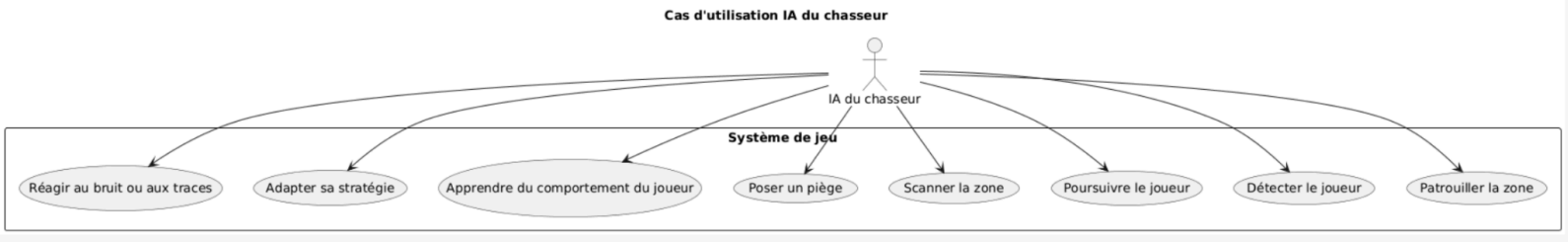
Se nourrir = Le joueur mange des plantes ou champignons pour restaurer sa jauge de faim ; certaines ressources procurent des effets bénéfiques tandis que d’autres, toxiques, représentent un risque.

Interagir avec l’environnement = Le joueur agit directement sur le monde : ramasser des ressources naturelles, creuser un terrier, déplacer un objet, ou masquer ses traces : chaque action influence le comportement du chasseur.

Créer une diversion = Le joueur détourne l’attention du chasseur en générant un bruit ou un mouvement à distance, par exemple en déclenchant un éboulis ou en déplaçant une branche.

Fabriquer un piège = En combinant des matériaux naturels comme des pierres, des branches et des lianes, le joueur conçoit de petits dispositifs destinés à ralentir ou tromper l’IA.

Utiliser une plante spéciale = En consommant certaines plantes rares, le joueur déclenche un effet temporaire comme une accélération, un camouflage ou une réduction du stress, au prix d’une prise de risque stratégique.



Patrouiller la zone = Le chasseur se déplace à travers la carte en suivant des points de patrouille définis ou calculés dynamiquement selon les zones d'activité du joueur.

Détecter le joueur = L'IA analyse son environnement via plusieurs capteurs simulés (sons, empreintes,mouvements). Lorsqu'un indice crédible est détecté, elle change d'état pour passer à la poursuite.

Poursuivre le joueur = Dès que le joueur est repéré, le chasseur active un mode de poursuite rapide : il suit la trace du lapin via un algorithme de pathfinding et garde une mémoire temporaire de sa dernière position connue.

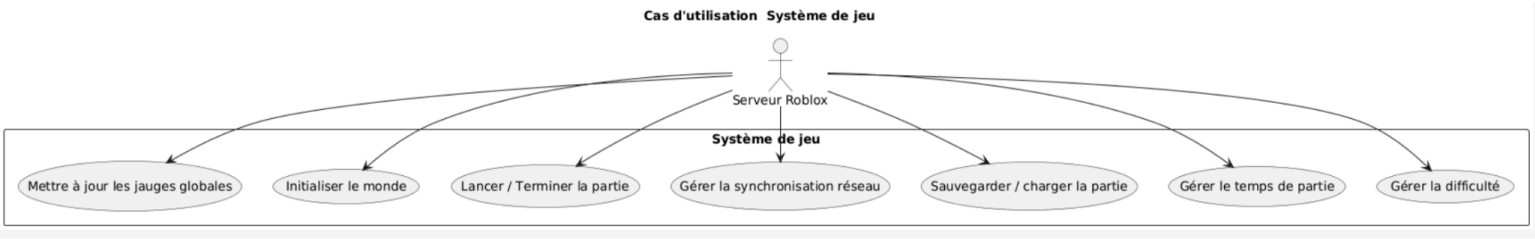
Scanner la zone = l'IA scanne visuellement ou auditivement les environs. Ce scan peut se faire de manière ciblée (autour d'un bruit) ou aléatoire (balayage périodique) le nombre de scan est limité.

Poser un piège = L'IA place des pièges dans des zones fréquentées par le joueur, souvent près des points de ressources ou des chemins de fuite connus. Ces pièges peuvent ralentir, ou piéger le joueur pendant quelques secondes.

Apprendre du comportement du joueur = Le chasseur enregistre les habitudes du joueur : ses itinéraires, ses cachettes, ses heures d'activité. Ces données sont utilisées pour affiner la patrouille et prioriser certaines zones.

Adapter sa stratégie = En fonction de ce qu'il apprend, le chasseur change sa manière de jouer : il peut réduire la distance entre ses rondes, piéger les zones souvent empruntées ou tendre une embuscade sur le trajet du joueur.

Réagir au bruit ou aux traces = Lorsqu'un bruit fort, une empreinte ou une action suspecte est détectée, le chasseur interrompt sa routine pour se diriger immédiatement vers la source. Si le joueur n'est pas trouvé, il mémorise l'événement pour renforcer la surveillance locale.



Gérer la difficulté

Le système adapte automatiquement la difficulté pendant la partie.

Par exemple, si le joueur s'en sort trop facilement, le chasseur devient plus agressif ou les ressources se font plus rares.

L'idée est de garder la tension du jeu sans le rendre injuste.

Gérer le temps de partie

Le jeu suit un cycle jour/nuit qui influence tout :

le comportement du chasseur, la visibilité, et les sons.

Le système gère aussi la durée de chaque cycle et les effets visuels ou sonores associés aux transitions.

Sauvegarder / charger la partie

Le système permet d'enregistrer l'état complet du jeu (position du joueur, IA, ressources, jauges, etc.).

Tout est stocké côté serveur pour qu'on puisse reprendre une partie exactement là où elle s'était arrêtée.

Gérer la synchronisation réseau

Ce module s'occupe de faire circuler les informations entre les joueurs et le serveur.

Chaque action (sons, déplacements, IA, météo) passe par le serveur avant d'être renvoyée à tout le monde.

Cela garantit que tous les joueurs voient la même chose au même moment.

Lancer / Terminer la partie

Au lancement, le système prépare le monde, installe le joueur et le chasseur, puis démarre la partie.

À la fin, il affiche les résultats et sauvegarde la progression selon que le joueur ait survécu ou non.

Initialiser le monde

Avant chaque partie, le système génère la carte, place les ressources et configure la météo.
C’est ce qui permet de créer des parties différentes à chaque fois.

Mettre à jour les jauges globales

Le système gère les jauges de faim, de stress, de bruit et d’énergie.
Elles changent selon ce que fait le joueur (courir, se cacher, creuser) ou selon la situation (proximité du chasseur, météo, fatigue).

Planning

Itération 1

Prototype jouable

Fonctionnalités à implémenter

- **Création des classes `Hunter` et `Lapin`**
    - Attributs : position, vitesse, santé, état.
    - Méthodes : Move(), TakeDamage(), Eat(), Attack().
    - Implémentation orientée objet en Lua (modules séparés).
- **Déplacement de base**
    - Gestion des inputs clavier pour le joueur.
    - Utilisation des services Roblox : UserInputService et Humanoid:MoveTo().
- **Système de faim et de vie du lapin**
    - Diminution progressive de la faim.
    - Mort du lapin si faim = 0.
    - Régénération partielle de la vie via la nourriture.
- **Attaque du chasseur**
    - Détection de collision donc portée de l'attaque.
    - Dégâts appliqués sur le lapin en cas de contact.
    - Animation d'attaque simple.
- **Nourriture du lapin**
    - Objets plantes ou carottes collectables.
    - Interaction via ProximityPrompt ou clic.
    - Réapparition automatique des ressources.
- **Menu de lancement de partie**
    - Interface basique : "Jouer / Quitter".
    - Initialisation de la scène et de l'apparition.
    - Réinitialisation du HUD.

Itération 2
Behaviour Tree (commencement itération 1)

Itération 3
Commencement machine learning

Itération 4
IA Lapin simple pour entraîner le hunter

Itération 5
Amélioration machine learning

Itération 6
Finition et amélioration