Projet Annuel

Expérimentation Bots pour un ou des jeux.

Tutoré par :

Florent Madelaine

Réalisé par :

Simon Catanese

Victor Descamps

Ethan Richard

30 Mai 2024

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL

IUT
SENART
FONTAINEBLEAU

Remerciements

Nous remercions Florent Madelaine, notre tuteur de projet, pour le temps qu'il a consacré à nos réunions et aux nombreuses pistes qu'il nous a proposées tout au long de notre travail. Nous le remercions aussi de nous avoir aidé à nous recentrer sur le projet de manière plus professionnelle.

Nous remercions également Dino, mascotte officieusement officielle du BUT Informatique de Fontainebleau, pour avoir généreusement accepté de figurer comme modèle d'avatar pour notre jeu.

Résumé

Dans le cadre du développement d'un jeu prévu pour comparer différents bots, ce rapport examine en détail le projet développé sous Unity mettant en lumière les fonctionnalités prévues et produites, les performances des bots et les aspects de notre travail.

En outre, le rapport montrera comment ce projet commencé tardivement et peu organisé s'est finalement concrétisé au prix de certains compromis fonctionnels. Les résultats mitigés de certains bots, nous amènerons également à une conclusion sur l'utilité des réseaux neuronaux dans un jeu peu complexe.

Table des Matières

Introduction

Le projet annuel de 3e année de BUT du groupe d'Ethan Richard, Simon Catanese et Victor Descamps consiste en un jeu complet développé sur le moteur Unity auquel peuvent participer différents bots et un joueur humain. Ce projet constitue le fil conducteur de notre année, nous permettant d'accomplir une tâche confiée par notre tuteur tout en ayant la liberté de choisir sa direction, ses objectifs, les logiciels utilisés, la méthode d'affrontement entre les bots, ainsi que de personnaliser des détails visuels selon nos préférences.

Nous avons choisi d'utiliser Unity en objectif personnel en raison de sa réputation dans l'industrie du jeu vidéo, ce qui nous aiderait dans le cas d'une poursuite de carrière dans ce secteur. L'intégration de bots était l'objectif principal qui a pour but de comparer les différentes méthodes de prise de décision et d'étudier les performances des différents types de joueurs, qu'ils soient humains ou machines, dans un jeu en temps réel peu complexe. Il est donc crucial pour nous de faire progresser ce projet en appliquant les connaissances acquises par des recherches approfondies, afin de créer un produit final qui reflète l'ensemble du travail fourni au cours de cette année.

Ce rapport présentera le cahier des charges et l'environnement de développement du projet puis passera en détail sur les différents aspects finaux du jeu avant de conclure sur nos observations.

1. Présentation

L'objectif du projet est de comparer l'efficacité de différents bots jouant à un jeu en temps réel. La décision collective a été de reproduire un jeu mobile populaire, Jetpack Joyride, sur Unity pour y faire se mesurer différentes variétés de bots et comparer leurs performances.

Les objectifs et contraintes spécifiques sont les suivants :

- Créer le Jeu, clone de Jetpack Joyride limiter à un certain nombre de fonctionnalités pour simplifier le travail des IAs[1].
- Implémenter un système permettant à des IA de prendre des décisions pour un joueur.
- Concevoir une architecture modulaire où chaque fonctionnalité du jeu est exposée par API [2].
- Mettre en place un système de collecte de données qui enregistre des informations pertinentes sur chaque partie jouée.
- Développer une interface permettant de visualiser et d'analyser les données collectées à but comparatif.
- Permettre la modification des règles d'une partie pour tester les performances des bots dans différentes conditions.
- Rendre le jeu moddable pour permettre l'implémentation de bots par un programme extérieur.

1.2 Présentation

La composante principale du projet est le jeu complet que nous avons développé. Celui-ci doit permettre à l'utilisateur de lancer des parties auxquelles des bots ou lui-même peuvent jouer. De nombreux paramètres permettent de modifier le déroulé d'une partie et la génération du niveau. Le jeu peut accueillir un nombre virtuellement illimité de joueurs dans une même partie afin de tester les performances de plusieurs bots en même temps. La présence d'un joueur humain amène certaines modifications visuelles tels qu'une transparence des autres joueurs et la disparition des obstacles ciblant uniquement les autres joueurs non-humains.

Le jeu fournit également un écran de statistiques permettant de comparer les joueurs entre eux, pour que l'utilisateur puisse étudier leurs performances.

---

[1] IA: Intelligence Artificiel

[2] API (Application Programming Interface) : une interface applicative qui permet de connecter le code externe avec l'architecture du projet

Des données structurées en JSON sont collectées tout au long des parties et sont sauvegardées localement sur l'ordinateur de l'utilisateur. Ces données sont exploitées par l'écran des statistiques.

2. Étude de l'état de l'art

Les choix techniques et technologiques du projet reposent sur une étude des solutions existantes. Cette étude nous permet de faire des choix judicieux répondant aux besoins du projet sans être excessif ou obsolète.

2.1 Développement du jeu

Développer un jeu en temps réel from scratch serait complexe et chronophage, alors que plusieurs solutions existent pour accélérer le développement d'un jeu et permettre une meilleure marge de manœuvre.

Unreal Engine est une des plateformes de développement de jeu les plus utilisées. Le moteur très puissant peut supporter de grosses charges graphiques, mais ce n'est pas l'aspect recherché par notre projet. UnrealEngine demande également plus de ressources à la machine l'exécutant et génère des fichiers de sortie plus gros.
En termes de développement, l'éditeur Unreal est plus complexe d'utilisation et les scripts peuvent être développés en C++, un langage très avancé, ou en NoCode[3] en reliant des boîtes logiques, ayant peu d'intérêt pour des étudiants en informatique. Unreal Engine n'est de ce fait pas le choix le plus intéressant.

Godot est un moteur de jeu open source plus léger avec un éditeur plus simple d'utilisation. Les scripts sont écrits en GDScript,un langage dérivé du python. Malheureusement, le moteur bénéficie de peu de ressources d'apprentissage par rapport aux autres plus populaires.

Unity est un autre moteur de jeu avec une popularité importante juste derrière Unreal. Il est très polyvalent et dispose du plus de ressources de documentation et tutoriels en lignes par rapport aux autres moteurs. La courbe d'apprentissage de l'éditeur est très accessible et les scripts utilisent le langage C# se rapprochant beaucoup du Java.
Unity dispose également d'une grande bibliothèque de plugins pouvant répondre à toute sorte de besoins. Notre choix porte donc sur Unity.

[3] NoCode: Programme informatique réalisé sans écriture de code. Cela consiste généralement à l'assemblage de boîtes logiques.

2.2 Plugins et outils

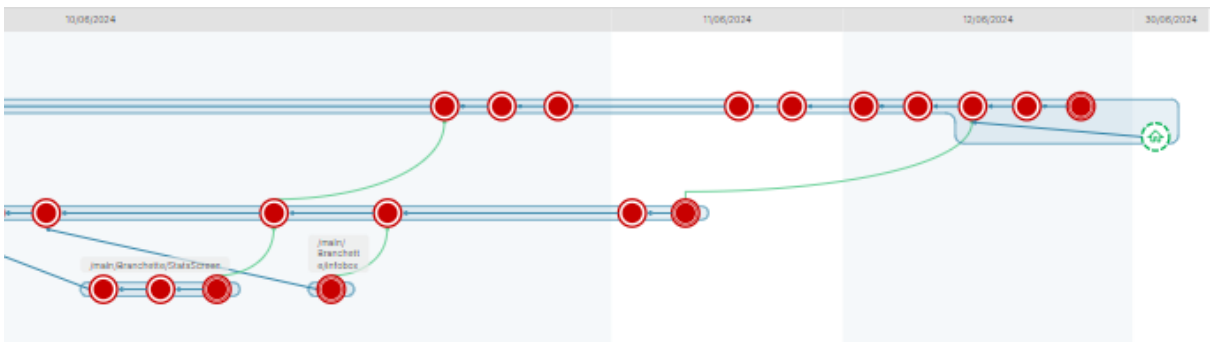
Unity possède une grande collection de plugins[4] pouvant accélérer le développement du jeu en prenant en charge des fonctionnalités. Lorsque le besoin est identifié, il est probable qu'un plugin puisse y répondre, évitant au développeur d'y consacrer trop de temps. Les deux plugins suivants ont permis de prendre en charge d'importants aspects du projet :

ML Agents est un plugin Unity permettant d'encadrer l'entraînement d'intelligences artificielles basées sur des réseaux neuronaux.
Cette bibliothèque offre un libre accès à de nombreux paramètres d'entraînement du nombre de neurones utilisés au facteur de prise de risque de l'IA. La bibliothèque nous donne des scripts Python préfet pour procéder à l'entraînement ce qui est idéal pour appréhender l'IA pour la première fois comme nous l'avons fait.

XCharts est un plugin permettant l'intégration de toute sorte de graphiques pour visualiser tout type de données. Il est très documenté et simple d'utilisation, et la grande diversité des types de graphiques qu'il propose correspond à notre ambition de pousser au maximum le traitement des données de jeu.

En dehors des plugins, un autre outil externe a grandement facilité le développement en groupe :

PlasticSCM est un client graphique utilisant Unity Version Control, le système de gestion de version officielle d'Unity. Les projets Unity n'étant pas supportés par Git, la gestion de version s'est entièrement déroulée avec Unity Version control. Combiné avec Plastic SCM, ce service possède les mêmes fonctionnalités que Git permettant notamment la gestion de branches, le contrôle de modifications et les marges.



Plugin: Un programme additionnel utilisé dans un logiciel ou sur une application internet

3. Conception du Projet

Un projet Unity comporte plusieurs répertoires autogérés par l'éditeur permettant la gestion des packages et des paramètres du projet. Le répertoire sur lequel travaillent les développeurs est le répertoire Assets, qui contient tous les fichiers et ressources utilisés dans le projet. Il contient notamment les scripts, les modèles, les textures, les animations, les scènes, etc.

Les différents type de ressources sont triés dans des sous-répertoires distincts. Exemple : les scripts se trouvent dans le répertoire Scripts, et les scripts en rapport avec les différents objets du jeu se trouvent dans le répertoire Scripts/GameObjectsScripts.

3.1 Architecture fonctionnelle

Chaque composante du jeu est gérée par un script unique suivant le design pattern Singleton :

- Les opérations concernant les joueurs d'une partie sont prises en charge par le script PlayersHandler
- La gestion du scores de chaque joueur et de l'affichage des écrans des score est faite depuis le script ScoreHandler
- La gestion d'une partie, contrôlant les différentes étapes au cours du déroulé du jeu, est assurée par le script GameHandler.
- La gestion d'une session, et des parties consécutives se fait dans le script GameController.

Ces scripts existant en instance unique sont responsables d'une composante particulière du jeu et les méthodes publiques qu'ils exposent permettent le déclenchement d'opérations spécifiques de n'importe où.

D'autres scripts sont chargés de parties plus spécifiques du jeu. Par exemple, PlayerScript est instancié sur chaque joueur d'une partie et agit de manière autonome pour demander une prise de décision sur le prochain mouvement à faire et réagir en cas de collision avec un autre élément du jeu.

3.2 Support de mods

L'intérêt d'un jeu supportant des bots personnalisés et offrant des moyens d'analyse de performance est l'opportunité de laisser l'utilisateur modder[5] le jeu pour y ajouter son propre bot.

Dans la prévision d'une telle éventualité, l'entièreté de l'architecture du code a été révisée . Les points à prendre en compte sont la communication d'informations à un programme externe et la gestion dynamique des joueurs pour qu'un joueur, personnalisé soit traité de la même manière par le jeu qu'un joueur natif.

3.2.1 : Modèle API et Singleton Pattern

En exposant de nombreuses méthodes publiques autorisant la collecte d'informations et délégant le déclenchement d'opérations spécifiques, le jeu offre un contrôle plus large de son fonctionnement et devient accessible au code externe.

Le fonctionnement du jeu est divisé en plusieurs parties toutes gérées par un script dédié suivant le design pattern Singleton. De cette manière, chaque script gérant une composante particulière du jeu est garanti d'exister en une seule instance continuellement accessible durant toute son exécution.

Une classe générique de base 'SingletonMB' permet d'implémenter par héritage un comportement Singleton de base à n'importe quel script, créant une référence statique à une instance unique lorsque celle-ci est appelée. Le suffixe "MB" de 'SingletonMB' est une abréviation de la classe 'MonoBehaviour' qui constitue la classe de base dont doit hériter tout script Unity afin d'être instancié dans le moteur.

PlayersHandler, ScoresHandler et LevelSpawner sont des exemples de scripts héritant de la classe 'SingletonMB'

3.2.2 Événements et Observer pattern

Un aspect fondamental d'un jeu moddable[6] est l'envoi régulier d'événements pouvant être écoutés par un script devant réagir à une étape particulière de l'exécution. Le moteur Unity traite les événements C# de manière synchrone sur le même thread d'exécution. Un événement

[5] Modder:Il s'agit du fait de modifier un objet ou logiciel.
[6] Moddable: adjectif mod comme modder

déclenché par un script appelle immédiatement les méthodes l'observant, dans leur ordre d'abonnement respectif.

En raison du contrôle limité du développeur sur l'ordre d'appel des méthodes, il est important qu'elles puissent toutes fonctionner indépendamment les unes des autres et qu'elles aient le même impact sur l'application peu importe leur ordre d'appel.

L'utilisation régulière d'événements permet aux programmes externes de recevoir des données particulières ou de réagir en déclenchant des opérations dans le jeu, par exemple en appelant le 'PlayersHandler' pour instancier un joueur personnalisé au moment de l'instanciation des autres joueurs.

3.2.3 Abstraction et prise en charge des joueurs

Afin que tout joueur instancié qu'il soit natif ou non, soit traité comme les autres joueurs par les systèmes du jeu automatiquement, ne laissant que le contrôle du joueur au script comportemental.

De cette manière, un utilisateur peut créer un nouveau joueur en concrétisant un joueur abstrait, lui définissant différents attributs comme une couleur, un nom et une nature (humain/bot). Il lui reste ensuite à étendre une classe comportementale et implémenter la méthode de prise de décision qui est appelée à chaque frame[7]. L'algorithme de prise de décision revient entièrement au joueur qui peut consulter le jeu via l'api ou les événements pour obtenir des informations, et choisir son moyen de traiter les données acquises pour prendre une décision de contrôle.
Un autre script peut également réagir à un événement au moment de l'instanciation des joueurs sélectionnés pour appeler la méthode d'instanciation en lui passant son joueur moddé pour l'intégrer au jeu.

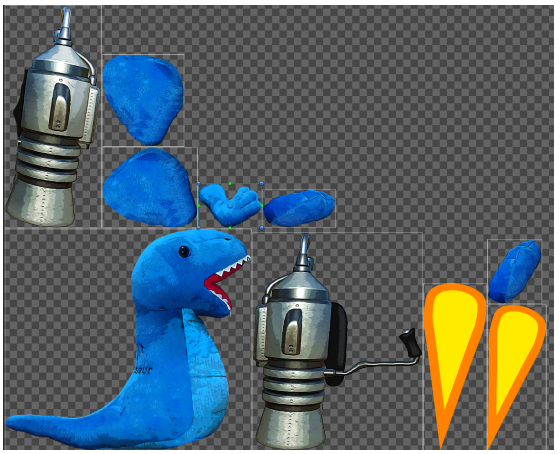
Le système du jeu prend en charge ce nouveau joueur en l'enregistrant dans la liste des joueurs instanciés et en lui associant un écran des scores. Les données de session concernant ce joueur sont inscrites à son nom.

[7] Frame:Une "image" qui représente l'ensemble des informations affiché graphiquement présent dans un "cadre"

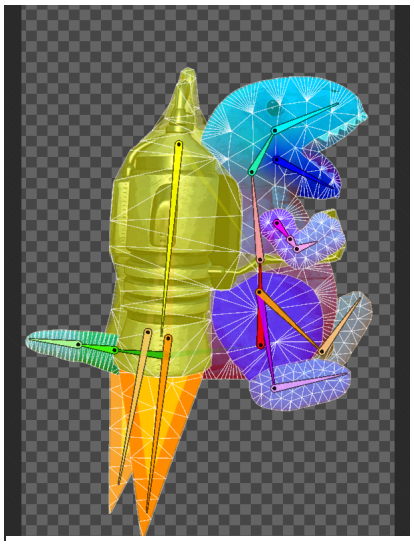
3.3 Expérience utilisateur

Un important travail visuel a été consacré au projet pour améliorer au mieux l'expérience de l'utilisateur. Du temps de travail a été réservé à la mise au point d'une texture d'arrière-plan, et d'un modèle animé de la mascotte officieusement officielle, Dino, ainsi qu'au développement de fonctionnalités améliorant le confort.

L'Avatar animé est réalisé à partir d'une photo de la peluche Dino. Les différentes parties de l'avatar (bras, jambes, corps, jetpack) ont été découpées, retravaillées et assemblées sur l'éditeur d'images Photoshop.



Visualisation de l'avatar désassemblé



Visualisation du squelette d'animation

L'avatar utilise 3 animations réalisées à la main. La réalisation de l'animation requiert en premier lieu la mise en place d'un squelette d'animation, contrôlant la position, la rotation et la déformation de chaque partie du corps de l'avatar.

Une animation est un ensemble de clés temporelles contrôlant la position, la taille et la rotation de chaque partie du squelette par intervalle de temps.

Ce travail de création est essentiel à l'immersion de l'utilisateur et donne une identité au jeu

D'autres détails esthétiques ont été réalisés pour raffiner l'interface et éviter un sentiment d'inconfort à l'utilisateur. Une police de caractère et un thème de couleurs donnent une identité visuelle à l'interface, et un arrière-plan en défilement infini permet de combler le vide et de dynamiser le menu principal.

Une bonne compréhension et une bonne clarté de l'interface sont importantes pour ne pas perdre l'utilisateur. L'écran de configuration de partie est équipé d'une infobox[8] affichant un texte descriptif pour chaque élément de l'interface survolée

[8] Infobox: Une zone du menu semblable à une "boite" ou sont situé les informations

par la souris. L'utilisateur peut ainsi connaître le rôle de chaque élément de l'interface.

Chaque type de joueur (humain/bot) a une couleur désignée, appliquée en filtre sur le jetpack du Dino lors de chaque partie. Ces couleurs permettent à l'utilisateur de mieux différencier les joueurs au cours d'une partie.
D'autres efforts ont été fournis pour ne pas gêner le joueur humain lorsqu'il joue en compagnie de bots : les autres joueurs deviennent transparents et les missiles ne ciblant pas le joueur humain sont masqués.

Une ligne d'arrivée est placée sur le niveau lorsque celui-ci est configuré avec une distance maximale définie pour clairement séparer la fin d'un essai et le début d'un autre essai.

4. Implémentation

- **Implémentation** : Expliquez comment vous avez développé chaque module, incluant les défis rencontrés et comment ils ont été résolus.
    - Le jeu (solution choisie pour spawn des laser)
    - le menu ?
    - les ia

4.1 Génération du niveau :

La génération des parties du niveau est entièrement procédurale et dépend d'une seed définie sur l'écran de configuration de la session de jeu. Seule la sélection des obstacles prédéfinis est influencée par cette seed.

Le script Singleton LevelSpawner est chargé de gérer l'instanciation et le cycle de vie des obstacles.

Les obstacles prédéfinis sont des assemblages manuels d'objets de niveau enregistré dans une série de Prefabs. On appelle cet assemblage d'obstacle un pattern.

À l'instanciation d'un pattern, sa longueur totale est calculée et sert de référence pour déterminer l'instant où il sera entièrement passé de l'autre côté de l'écran. Un nouveau pattern est alors instancié à l'emplacement d'apparition et le pattern précédent est disposé. De cette manière, il est garanti qu'un écart vide large d'au moins un écran entre chaque pattern permettra au joueur de récupérer une trajectoire optimale après sa sortie du dernier obstacle.

Les différents objets pouvant figurer dans les patterns sont les lasers, les pièces bonus et les pièces malus.

4.1.1 Les lasers

Les lasers sont les obstacles les plus courants du jeu, infligeant une diminution conséquente du score du joueur lorsqu'il entre en contact avec eux.

L'objet concret représentant un laser est l'ensemble de 3 sous-objets : deux points et un segment. Les deux points sont placés par le développeur aux bouts désirés du laser, et la position du segment est calculée à chaque frame du jeu.

4.1.2 Les pièces

Les pièces bonus et malus provoquent respectivement une légère augmentation ou diminution du score lorsqu'elles sont touchées par un joueur. Leur

emplacement demande parfois au joueur de prendre des risques en s'approchant de lasers ou en effectuant une manœuvre d'évitement risquée.

Cette mécanique est pensée pour départager plusieurs joueurs simultanés ayant réussi un niveau en récompensant la prise de risque.

4.1.3 Les missiles

Les missiles sont des obstacles périodiques. Lorsque le joueur traverse deux séries de patterns d'obstacles, une alerte de 5 secondes clignote à droite de l'écran en suivant la hauteur du joueur. Au bout des 5 secondes, un missile est instancié à droite de l'écran et à la hauteur actuelle du joueur.

Le missile avance horizontalement à une vitesse fixe additionnée avec la vitesse de défilement du niveau. Il inflige une grande diminution de score s'il entre en contact avec le joueur qu'il cible.

Un missile doit apparaître pour chaque joueur présent dans la partie et affecter seulement le score du joueur ciblé. Pour faire fonctionner ce système, le script MissileSpawner gérant le cycle de vie d'un missile est instancié avec chaque joueur dans la partie. De cette manière, chaque joueur est traité par un MissileSpawner indépendant des autres.

4.2 Paramètres et règles :

La mise en place de règles diverses et personnalisables s'est avérée nécessaire pour permettre à l'utilisateur de tester ses capacités ou celles des bots dans différentes conditions. L'écran de configuration d'une partie permet d'activer ou désactiver l'apparition de bonus ou d'obstacles dans un niveau, choisir la seed[9], spécifier un nombre d'essais consécutifs, un temps maximal à sa session de jeu, et une distance à atteindre pour les joueurs à chaque essai.

Les préférences du joueur sont mémorisées et récupérées à l'initialisation des niveaux.

L'usage de malus/bonus peut servir à améliorer la précision des comparaisons entre les bots et/ou joueur humain via d'autres facteurs que la survie.

Écran des statistiques :

Afin d'étudier l'efficacité des bots entre eux, et comparés à un humain, l'écran des statistiques a pour vocation de fournir une multitude de graphiques divers exploitant les données collectées durant chaque partie. Le temps consacré à cette interface ne nous a permis que d'implémenter une étude de l'évolution du score des différents joueurs au cours des 4 premières parties d'une session sélectionnée par l'utilisateur.

[9] Seed: Ensemble de chiffre représentant la génération des obstacles sur la partie

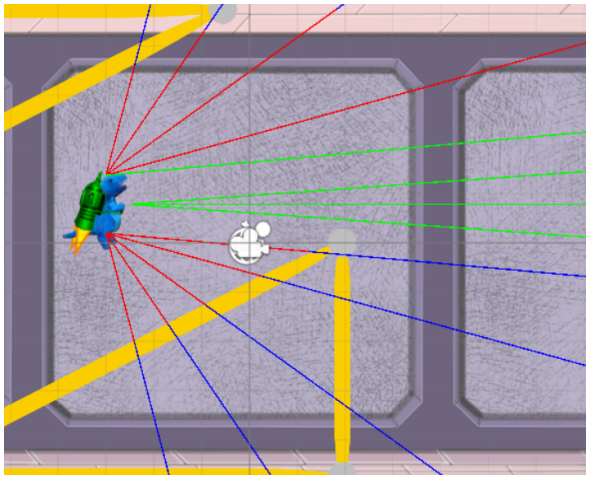
Les graphiques sont basés sur la bibliothèque XCharts permettant de générer toutes sortes de graphiques aisément.

5. Les Intelligences Artificielles

Pour la création des IA comme dis plus haut, nous avons utilisé la bibliothèque ML Agent nous permettant d'associer un script comportemental Unity à un réseau neuronal de type ONNX (Open Neural Network Exchange, un format polyvalent pour frameworks d'apprentissage) et de procéder à son entraînement grâce à des script python associés.

5.1 L'Observateur

L'Observateur a été conçu pour être l'IA principale du projet. Elle consiste en un réseau neuronal composé de trois couches de 128 neurones chacune en plus d'une sortie booléenne et 29 entrées. Ces entrées représentent la position verticale de dino, sa vélocité horizontale et verticale, ainsi que 13 couples (distance et nature de l'objet détecté) illustrant la vision de l'IA. Cette vision est simulée par des lasers projetés dans de nombreuses directions, comme illustrées ci-dessous.



Visualisation des Raycasts de l'observateur

5.1.1 Configuration Réseau neuronal

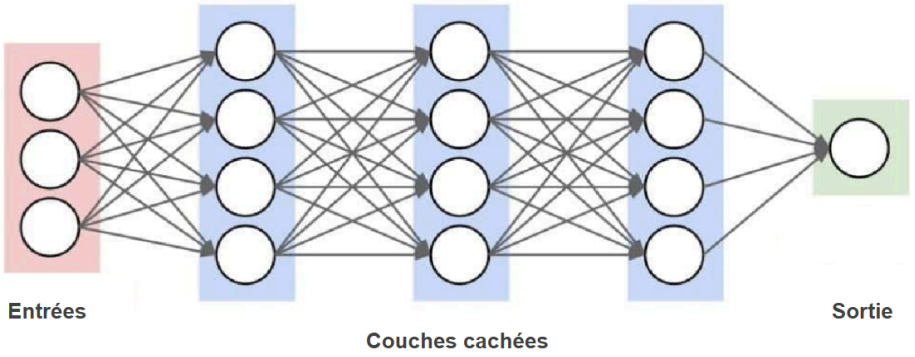
Plusieurs configurations de neurones ont été testées, telles que 2 couches de 128 neurones et 2 couches de 64 neurones. Cependant, nous avons décidé de retenir la configuration de 3 couches de 128 neurones, car elle offrait un apprentissage aussi rapide et généralement plus efficace, que les réseaux plus petits.

5.1.2 Configuration Raycast[10]

Plusieurs configurations de lasers ont été envisagées et testées : partant toutes d'un même point, dans des directions différentes, ou n'étant simplement pas centrées dynamiquement sur Dino. Toutefois, seule celle-ci a été retenue, car nous ne voulions pas créer différents bots trop similaires, et cette configuration offrait les meilleurs résultats. Les angles des raycasts sont les suivants : `[75°, 55°, 35°, 15°, 5°, 4°, 0°, 4°, 5°, 15°, 35°, 55°, 75°]` lus du plus haut au plus bas. À noter que les cinq premiers partent de la tête de Dino, les trois centraux du milieu de son corps, et les cinq derniers de ses pieds.

Les raycasts sont propres au personnage et non à l'IA et sont donc accessibles par le biais d'une API, nous avons fait ce choix ici et ailleurs afin d'assurer une grande modularité à notre jeu.

5.1.3 Illustration du réseau neuronal utilisé



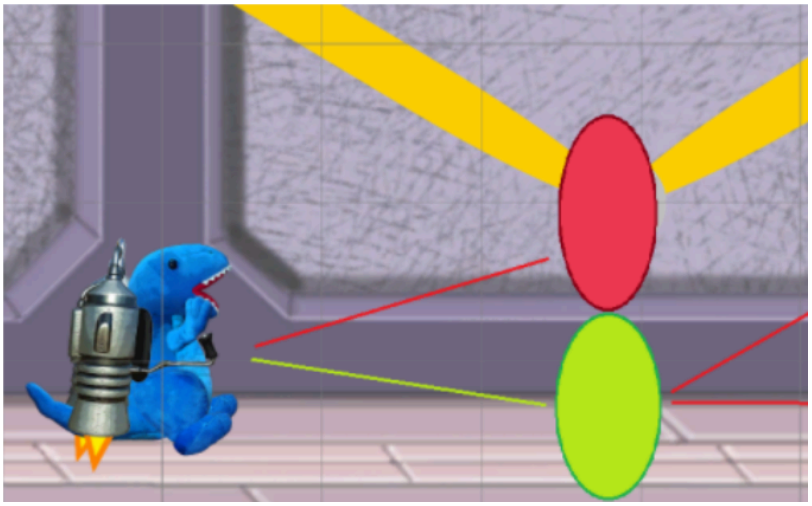
Modélisation du réseau neuronal utilisé : la partie rouge est en réalité composée de 29 neurones, et la partie bleue de 128 neurones par couche.

[10] Raycast : Technique d'acquisition d'information via l'envoi d'un "rayon" à partir d'un point vers une direction particulière.

5.2 L'Itérateur

5.2.1 Conception Initiale

L'Itérateur a été conçu pour être une IA algorithmique qui utilise un arbre de possibilités et une logique similaire à celle de l'algorithme Min-Max pour choisir le meilleur chemin. Cette approche permet d'explorer différentes options et de sélectionner la trajectoire optimale pour atteindre un objectif ou éviter des obstacles. Comme illustré ci-dessous.



Visualisation du principe d'exploration des chemins :
Vert = chemin valide ; Rouge = chemin dangereux

5.2.2 Nouvelle Approche

En raison de contraintes de temps, une autre méthode pour l'IA a été envisagée. Cette nouvelle approche utilise la même API que l'Observateur pour accéder à des capteurs raycasts. L'Itérateur collecte les distances des rayons projetés et les combine en appliquant des poids arbitraires. Cette combinaison permet de déterminer s'il doit monter ou descendre pour éviter les dangers les plus proches.

5.2.3 Pondération des Rayons

Après avoir expérimenté avec plusieurs combinaisons de poids pour les rayons, les valeurs suivantes ont été retenues : `[3f, 5f, 5f, 3f, 12f, 25f, 0f, 25f, 12f, 3f, 5f, 5f, 3f]`. Ces poids sont appliqués aux rayons de haut en bas, chaque rayon représentant une direction spécifique. Ils influencent le comportement de l'Itérateur en lui indiquant quelle direction suivre pour s'éloigner des obstacles.
Étant donné que monter est légèrement plus rapide que descendre, il aurait été logiquement plus efficace de donner un poids plus léger aux capteurs situés vers le bas. Cependant, lors des tests, nous n'avons pas observé de différence significative de performance en utilisant cette approche. Par conséquent, pour des raisons de simplicité, une combinaison symétrique a été adoptée. On peut observer que le poids du raycast central est nul car cette IA repose sur une stratégie d'évitement basée sur la peur des lasers. Attribuer un poids au raycast central n'aurait pas aidé à prendre la décision de monter ou descendre. Nous n'avons pas constaté que le manque de vision frontale posait problème, cette vision étant compensée par les raycasts adjacents.

5.2.4 Logique de l'Algorithme

L'algorithme de l'Itérateur peut être grossièrement représenté par le pseudo-code suivant. Ce code montre comment l'Itérateur utilise les informations des rayons pour prendre une décision sur la direction à prendre.

```
deplacement_total = 0

pour chaque Rayon :

    force_eloignement = calculer_direction_eloignement(distance)

    poids = obtenir_poids_du_rayon(rayon)

    deplacement_local = force_eloignement * poids

    deplacement_total += deplacement_local


si deplacement_total > 0:
```

```
    voler_vers_haut()
```

5.3 L’Omniscient

5.3.1 Conception Initiale

L'IA Omnisciente était une proposition ambitieuse dans notre projet, visant à utiliser un réseau neuronal pour collecter toutes les informations disponibles sur le terrain : les coordonnées des lasers, des pièces, des missiles, etc. Cette IA devait analyser ces données pour prendre des décisions éclairées et optimiser ses performances sur le terrain.

5.3.2 Compromis

Cependant, en raison de la complexité de l'apprentissage associée à ce modèle et du manque de temps, nous avons choisi de concentrer nos efforts sur d'autres IA plus réalisables. Pour L’instant, le bot Omniscient peut donc être sélectionné, mais il jouerait aléatoirement en attente de l'implémentation d’une IA plus réalisable.

Il est néanmoins toujours intéressant de pouvoir mettre en compétition des joueurs et des IA contre une IA de niveau 0.

6. Résultats

Actuellement, nous avons développé un menu qui permet de choisir et de définir divers réglages pour comparer les IA dans des conditions variées.

Nous sommes satisfaits de notre système de génération de niveaux, qui est entièrement procédural et dépend des configurations définies à partir de l'écran de configuration. Notre menu permet de sélectionner et de tirer des patterns d'obstacles que nous avons préalablement créés. L'apparition d'événements tels que les missiles, les bonus et les malus dans les patterns est bien intégrée dans la version actuelle et fonctionne à merveille.

Nous avons réussi à intégrer deux bots fonctionnels et un bot aléatoire, bien que le manque de temps nous ait empêchés d'en développer plus. Nous avons également ajouté la possibilité pour l'utilisateur de jouer au jeu.

Cela diffère quelque peu des objectifs initiaux, qui visaient à créer trois à quatre bots fonctionnels. Néanmoins, les résultats obtenus sont concluants et s'approchent en grande partie de l'objectif principal d'expérimenter des bots pour un ou plusieurs jeux, actuellement en utilisant deux bots ayant des approches significativement différentes.

6.1 Un test de déterminisme

Le déterminisme d'une partie est important pour qu'un résultat ne dépende pas de facteurs non contrôlés par le programme. Un test fiable permettant de vérifier le déterminisme du jeu consiste à faire parcourir le même niveau plusieurs fois au bot algorithmique, l'itérateur.
Nous pouvons conclure que le jeu est déterministe si 4 parties ayant les mêmes configurations ont un déroulement identique. Cette vérification est possible grâce à l'écran des statistiques permettant d'analyser l'évolution du score du joueur dans le temps.
Si pour les 4 parties réalisées, les 4 graphiques son identiques alors nous pouvont conclure que le jeu est déterministe.

Le résultat observé démontre cependant qu'aucune des 4 parties ne s'est réalisée de la même manière qu'une autre. Nous pouvons donc conclure que le jeu n'est pas déterministe à l'heure actuelle.

6.1.1 Cause

Ce non déterminisme est probablement dû à la variable temps entre chaque frame qui est un facteur de chaque élément évoluant en temps réel à des fins de synchronisation. En d'autres termes, si le jeu avance globalement à une vitesse constante, l'intervalle de temps entre les frames est variable. En rejouant la même

partie plusieurs fois de suite, il y aura inévitablement un décalage du joueur à la frame x de cette partie par rapport à la frame x de la partie précédente.

C’est là une démonstration parfaite de la théorie du chaos engendrée par l’aspect temps réel du jeu.
On pourrait se soustraire de ce chaos en se séparant de l’aspect temps réel du jeu, afin d’assurer un intervalle fixe entre chaque frame. Des ralentissements du jeu engendreraient donc un retard de l’horloge interne du jeu par rapport à l’horloge réelle, mais le jeu deviendrait entièrement déterministe dans le cas d’un bot algorithmique tel que l’itérateur actuel.

7. Discussion

Au vu de la popularité du jeu Jetpack Joyride, il existe d'autres travaux d'implémentations d'IA sur Internet, par exemple sur la chaîne JTexpo, une chaîne YouTube anglaise qui explore différents algorithmes spécialement dans le domaine de l'IA, ou où le créateur a recréé le jeu avec Python pour y entraîner une IA neuronale, vidéo disponible ici. Par ailleurs, l'IA reprend une approche proche de celle de notre vidéo omnisciente.

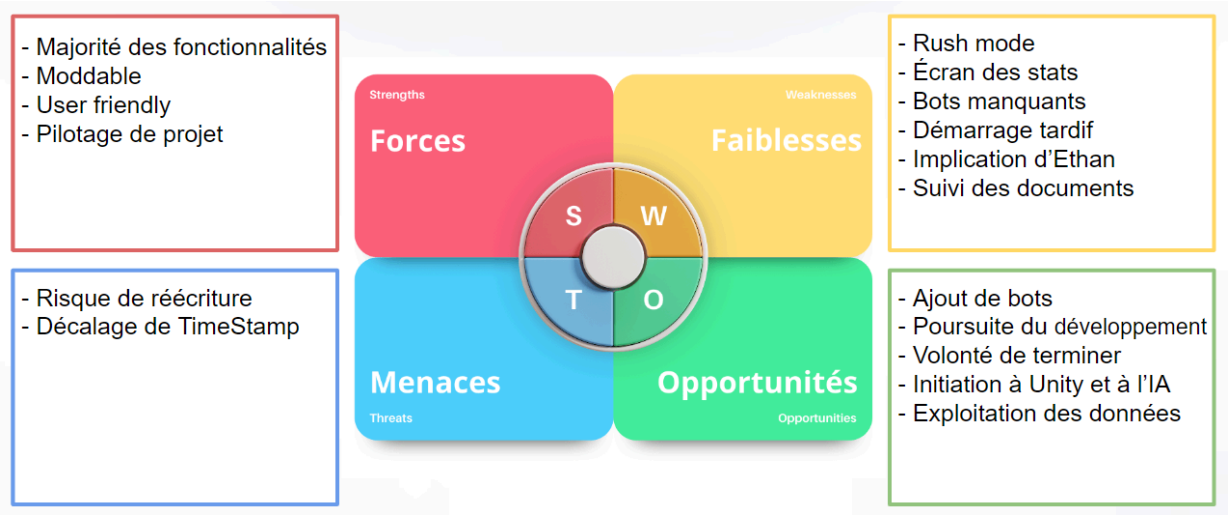
Aussi, nous pouvons trouver des usages des méthodes d'apprentissage et d'utilisation des IA sur d'autres cas de figure par exemple dans le jeu trackmania (un jeu de course automobile) ou la création de véhicules contrôlés par IA afin de battre des joueurs aguerri à été plus d'une fois testé. Nous vous proposons l'une d'elles ou l'utilisateur nommé Yosh à développé une IA sur Trackmania à base d'apprentissage par renforcement afin de développer les performances et les résultats de l'IA

Voici la vidéo : https://www.youtube.com/watch?v=Dw3BZ6O_8LY
Par ailleurs, c'est sur ce jeu que l'idée de l'usage des raycasts fut tirée ou une revue expliquait le développement aussi d'une IA acquérant les informations de son environnement via plusieurs raycasts et l'usage d'un réseau neuronal .

Retours d'expérience

Malgré les défis rencontrés tout au long de l'avancement du projet, cette expérience a été un très bon moyen d'accroître nos capacités de travail en groupe. Elle nous a également introduits de manière approfondie dans les domaines de l'intelligence artificielle et de la création de jeux vidéo, deux domaines qui nous intéressent énormément.

Diagramme SWOT portant sur le déroulement du projet



8. Conclusion

8.1 Résumé des réalisations

Nous avons mis au point un système robuste de génération procédurale de niveaux pour notre jeu, permettant une configuration flexible et diversifiée grâce à un menu intuitif. Les principales fonctionnalités incluent :

- Un menu de sélection et de définition des réglages pour comparer différentes IA dans des conditions variées.
- Une génération procédurale des niveaux basée sur les configurations définies par l'utilisateur.
- Une intégration réussie de patterns d'obstacles pré-créés, avec des événements dynamiques tels que des missiles, des bonus et des malus.
- Le développement et l'intégration de deux bots fonctionnels, ainsi qu'un bot aléatoire.
- La possibilité pour l'utilisateur de jouer au jeu, enrichissant ainsi l'expérience utilisateur.

8.2 Impact et implications

Les résultats obtenus démontrent la faisabilité et l'efficacité de notre approche procédurale pour la génération de niveaux dans un jeu. En offrant une grande flexibilité de configuration et en intégrant divers événements dynamiques, notre système enrichit considérablement le gameplay et permet une évaluation précise des performances des différentes IA dans des conditions variées. L'intégration de bots fonctionnels et la possibilité pour les utilisateurs de jouer renforcent l'interactivité et l'engagement des joueurs. Cette approche non seulement diversifie les défis pour les IA et les joueurs, mais aussi permet d'identifier et de comparer les stratégies de manière approfondie, contribuant ainsi à des développements futurs plus sophistiqués et adaptés.

8.3 Perspectives futures

Pour améliorer et étendre notre projet, nous envisageons les développements suivants :

- Développement de nouveaux bots : Ajouter plus de bots avec des comportements et stratégies variées pour enrichir les comparaisons et analyses.
- Amélioration de l’apprentissage automatique : Implémenter des mécanismes d'apprentissage automatique pour améliorer les performances des bots et adapter leurs stratégies en temps réel.
- Envisager une amélioration afin d'affiner l'ergonomie et l'intuitivité du menu de configuration pour une expérience utilisateur encore meilleure.

En conclusion, notre système actuel offre une base solide pour l'évaluation des IA dans des environnements de jeu variés. Les perspectives futures visent à enrichir cette base pour créer une expérience encore plus immersive et diversifiée.

Bibliographie

La chaîne anglaise Code Monkey qui nous à permis d'en apprendre plus sur le langage C# et un peu Unity afin d'aider au projet en ayant des bases communes .

chaine Youtube : https://www.youtube.com/@CodeMonkeyUnity

Nous avons trouvé des réponses et des corrections grâce à l'aide de plusieurs forums et sites où des personnes partagent leurs connaissances sur des sujets proches de nos besoins.

Nos principales sources d'information incluent les forums et la documentation de Unity, la documentation de Plastic SCM, les tutoriels sur UI Builder et la documentation de ML-Agents.

Documentation unity : https://docs.unity.com

Documentation Ml-Agents :

https://unity-technologies.github.io/ml-agents/ML-Agents-Toolkit-Documentation/