This Was Fine

Projet étudiant

ICAN

3GD

2023/2024

Au menu d'aujourd'hui :

Introduction

Game System

Player Motivation

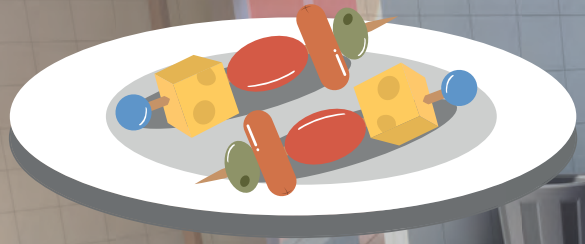
Level Design

Charte Graphique

Charte Sonore

Programmation

Introduction



Entrées

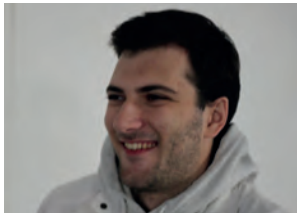
Introduction

Équipe

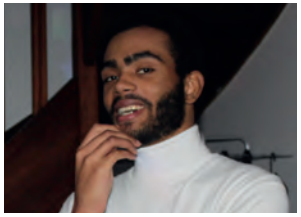
Berthier Sébastien
Concept Art by IA
Direction Artistique visuelle

Coustenoble Antoine
Lead Direction Artistique visuelle
Game Artist
Documentation

Delzangles Paul
Sound Designer
Documentation

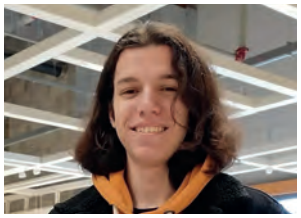


Denis Bastien
Lead Game Design
Level Design
Documentation



Helderald Matthias
Vfx Artist
Direction artistique visuelle

Limon Théo
Game Programer
Documentation technique



Marnet Matthieu
Level Designer
Game Design

Introduction au sujet

Dans le cadre de notre 3ème année de Bachelor Game Design à l'ICAN, nous devons réaliser un projet annuel sur un thème imposé. Le projet se découpe principalement en deux semestres avec des objectifs bien définis :

Le premier semestre doit servir à réaliser un proof of concept, le deuxième consistant à réaliser une vertical slice de notre projet.

Cette année, le thème du projet est le premier épisode d'un triptyque de l'artiste *Grégory Chatonsky* nommé *"La ville qui n'existait pas"*. Ce premier épisode se nomme *"Les espaces latents"* et se constitue d'une collection de 25 images affichées sur 25 façades d'immeubles de la ville du Havre.

Ces fresques représentent une version alternative du Havre entre 1895 et 1944 avant et après sa destruction. Ces images sont des collages réalisés grâce à une intelligence artificielle générative.

Ce modèle d'IA a reçu un apprentissage spécifique alimenté par le fond d'archives photographiques de la ville du Havre, des prises de vues documentaires sur place et de récits sur l'histoire de chaque lieu.

Le rapport au temps dans ces œuvres fût pour notre groupe un fil directeur, voir ces collages de la ville sous différents états de construction et de destruction nous a particulièrement inspirés.

Introduction

Processus Créatif

Une fois le sujet en notre connaissance, nous avons fait une première réunion pour discuter de tout ce à quoi nous faisait penser cette œuvre.

Une fois toutes les idées mises à plat, nous nous sommes donnés une semaine pendant laquelle tout le monde développait rapidement un prototype chacun de son côté. Au bout de cette semaine, nous avons fait une deuxième réunion pour voir tous les concepts et décider sur quel concept nous partirions.

Cette façon de faire nous a permis de voir plusieurs possibilités et a permis à chacun de proposer une idée. Nous nous sommes donc retrouvés avec un certain nombre de concepts. Après plusieurs phases de réflexion et de discussion, nous avons pu isoler deux prototypes :

- Premier prototype : Temps

Le premier prototype était un plan en 2D dans lequel des objets dotés de physique pouvaient rebondir, s'entrechoquer. Le joueur avait la possibilité de faire revenir le temps en arrière pour visionner les mouvements des objets de manière inversée.

L'idée de ce prototype nous vient de l'utilisation du temps dans les œuvres qui composent le sujet. En effet, *Chatonsky* joue avec les époques et semble à sa manière remonter le temps grâce à ses collages.

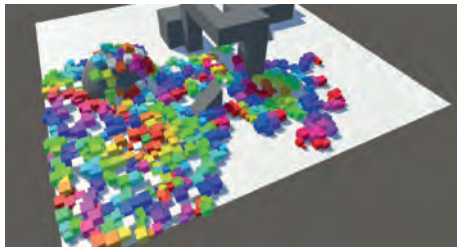
- Second prototype : Blob

Le deuxième prototype était un jouet dans lequel une masse informe recouvrait peu à peu tout l'espace de jeu. Le joueur pouvait effacer, couper des parties de ce Blob pour ralentir sa progression, le séparer et observer son comportement.

Ce prototype quant à lui est issu d'une interprétation toute autre du sujet, plus orienté sur ce qui se passe au niveau des masses violettes qui semblent envahir les œuvres de l'artiste. Ces sortes de blobs violets nous ont fait penser à une forme de vie pour le moins insolite qui est devenue populaire il y a quelques années, le blob. Nous avons donc essayé de reproduire le comportement de ces blobs pour voir ce que cela pourrait donner une fois en jeu.

Proto n°1

Ci-dessus une image du premier prototype, ici les tracés représentent le chemin qui, à ce moment, à déjà été réalisé par les balles.



Proto n°2

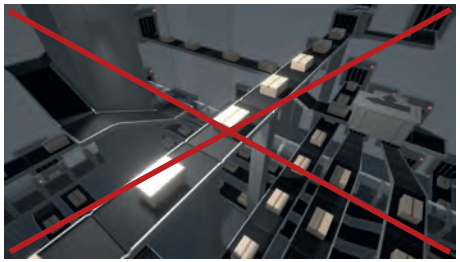
Ici le blob se répend dans l'espace de jeu et recouvre petit à petit les différents éléments.

Choix et Intentions

Au premier semestre, nous avions donc choisi le prototype n°1, la fantasy du temps nous avait séduite et nous avons pu grandement expérimenter à son sujet. Malheureusement, nous n'avons pas su correctement exploiter notre noyau et suite aux soutenances de fin de semestre, nous avons décidé de revenir un peu en arrière. Nous avons donc épuré le plus possible notre gameplay et nous avons cherché une nouvelle façon d'exploiter notre noyau système.

Nous avons vite opté pour quelque chose que nous avions mis de côté par le passé, l'utilisation de personnages non joueurs régis par des ia. Nous avions pour idée un environnement avec un système d'embranchement. Le but aurait été de découvrir les actions de pnj spéciaux dans un environnement fermé tout en s'amusant à changer le déroulement des choses en les bloquant dans leurs actions. Ce système, qui comprenait notre feature de contrôle temporel s'est avérée trop ambitieuse alors il nous a fallu faire un choix.

Nous avons tranché dans le vif et décidé de retirer le contrôle du temps de notre gameplay, qui, grâce aux IA était déjà bien assez divertissant. Le simple fait de déranger les personnages et d'aller à l'encontre de leurs volontés tout en observant ensuite leurs réactions s'est avéré très divertissant et amusant, c'est donc pourquoi nous avons fini par en faire le cœur de notre système.



Intentions :

- Nous voulons proposer aux joueurs un écosystème capable de s'auto réguler et d'absorber les actions réalisées par ces mêmes joueurs.

- On souhaite pousser le joueur à expérimenter en lui proposant une expérience bac à sable avec la possibilité d'amener l'environnement jusqu'à plusieurs états différents en fonction des actions réalisées avec les personnages dans l'environnement.

- Nous avons comme objectif de créer des sessions de jeu courtes permettant au joueur de tester toutes sortes de stratégies sans pour autant avoir besoin d'attendre que la situation de jeu précédente se résorbe grâce au système.

Fiche d’identité

Pitch : This Was Fine est un jeu sandbox donnant aux joueurs l’opportunité d’interagir avec petits personnages évoluant dans un environnement donné.

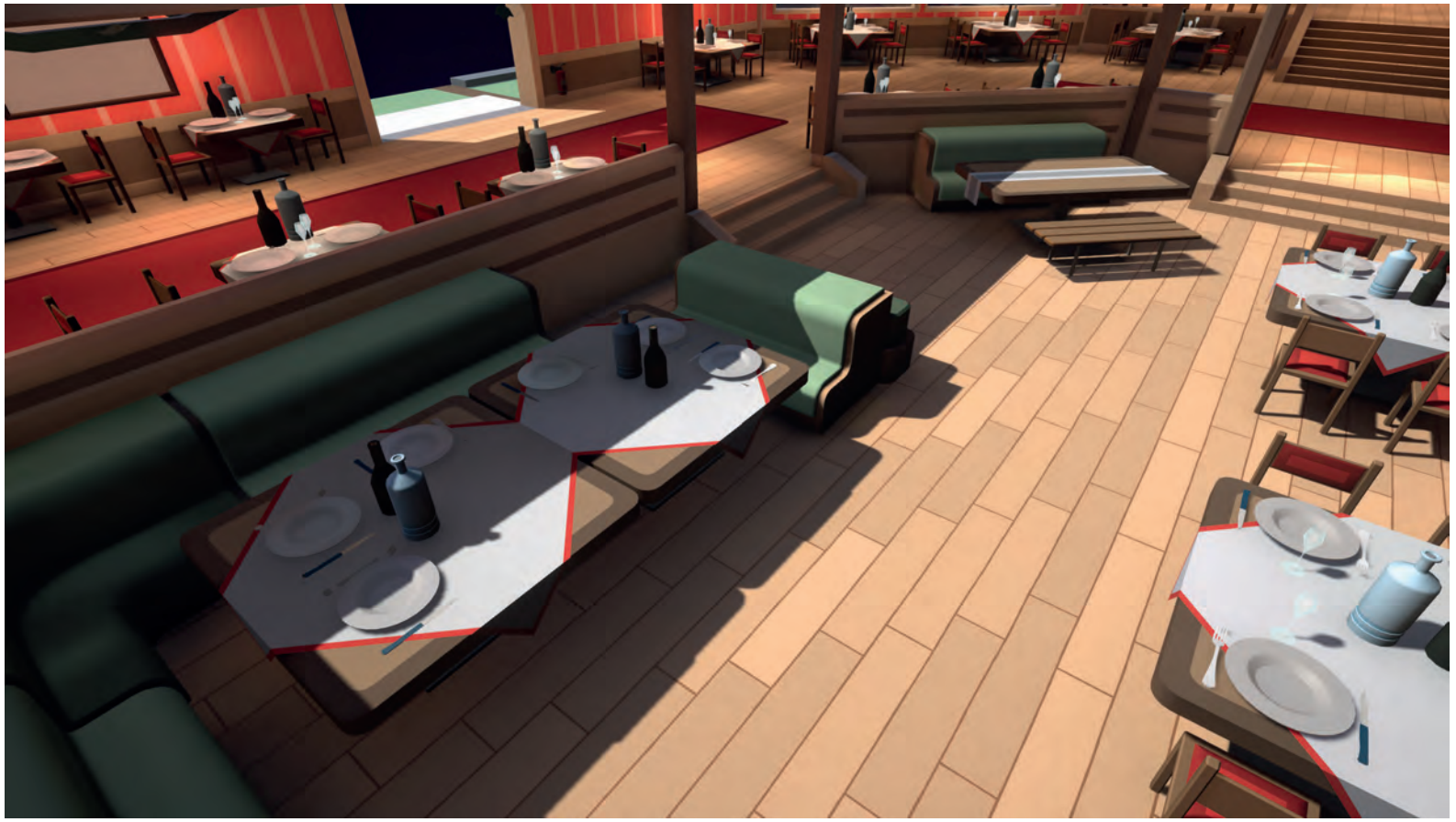
Fantasme : This was fine permet aux joueurs d’interagir avec un écosystème autosuffisant et capable de se réguler de lui-même.

Support : This was fine est un jeu développé pour PC et est fait pour se jouer au clavier/ souris.

Cible : This Was Fine permet aux joueurs de réaliser toutes sortes d’expériences dans un environnement remplis de petits personnages.

USP : This Was Fine permet aux joueurs de réaliser toutes sortes d’expériences dans un environnement remplis de petits personnages.

KSP : This Was Fine est facile à prendre en main grâce à ses contrôles orientés sur la souris et son interface intradiégétique.



Références Game Design

Tout au long du projet, nos références ont évolué, nous avons trouvé de nouvelles références, certaines références déjà présentes devenaient obsolètes au cours des différentes évolutions du projet. Suite aux divers changements opérés sur le projet, nos références gameplay ont drastiquement changées, en voici donc quelques-unes :

Untitled Goose Game

Ce jeu est pour nous une référence majeure. Nous nous inspirons beaucoup du rôle de perturbateur assigné au joueur. Le fait de donner au joueur le pouvoir d'embêter son monde et de faire comme bon lui semble nous a séduit, c'est pourquoi s'en est devenu le centre de notre gameplay.

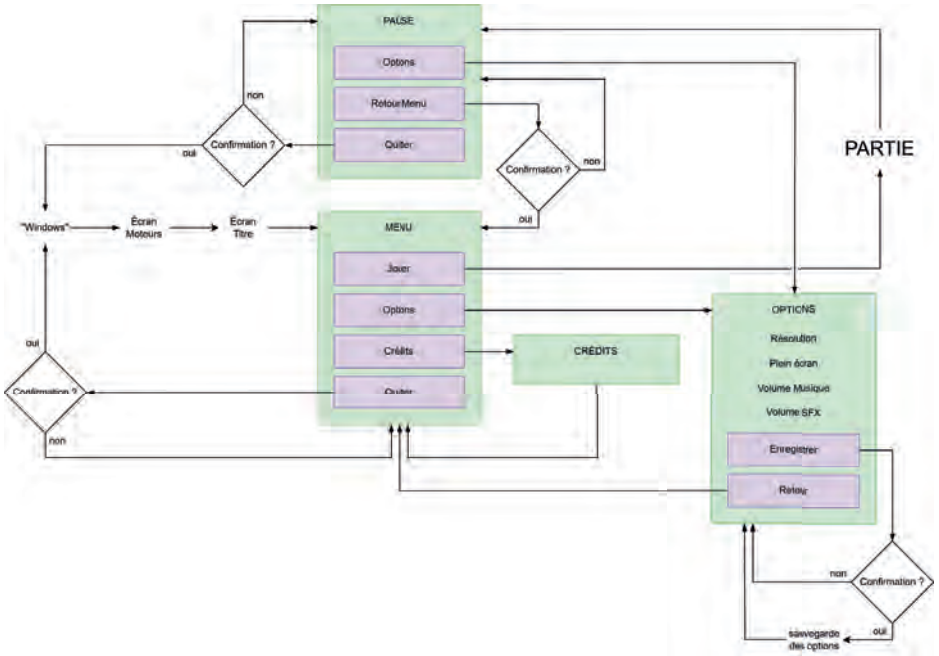
Les Sims

La série des jeux Sims constitue également une grande source d'inspiration, surtout pour nos personnages non joueurs. Nos NPC ont tout comme dans les Sims des barres (cachées pour notre jeu) de besoins les incitant à réaliser différentes actions en fonctions de l'état des différentes barres.

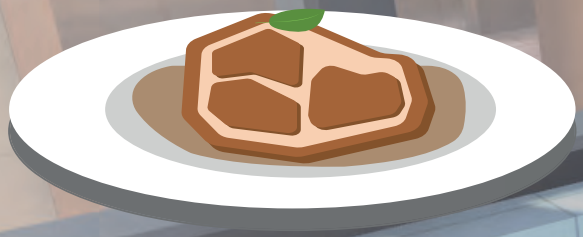
Overcooked 2

Notre jeu se déroulant dans l'environnement d'un restaurant, on ne pouvait que penser à lui. Nous nous en inspirons en partie pour le côté maladroit des personnages. Ce jeu est aussi une grande inspiration pour notre direction artistique, que ça soit pour l'environnement, les props, mais aussi les sons.

Flow Chart



Game System



Viandes

3C's

Noyau et tensions

Mécaniques

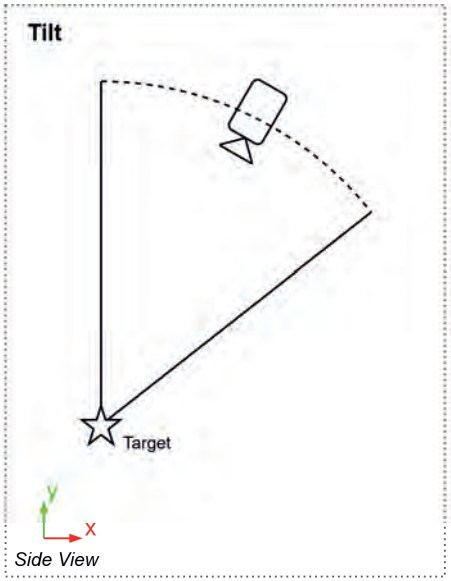
Boucles

Ventrices

Camera

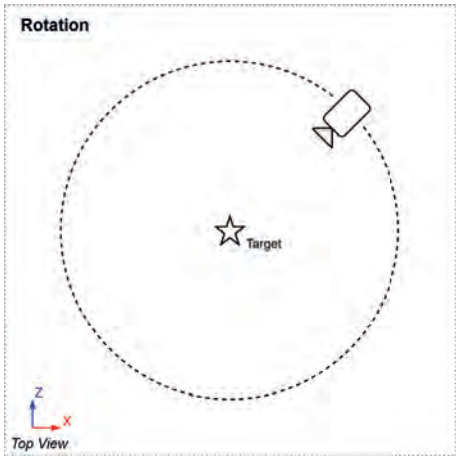
Nous proposons un jeu de style sandbox dans lequel le joueur doit pouvoir survoler et inspecter l'environnement à la fois de façon macro et micro. Pour cela, notre caméra reprend un principe de caméra en god view, en vue du dessus. Le joueur peut déplacer la caméra et la faire tourner sur point de pivot situé au niveau du sol de l'environnement.

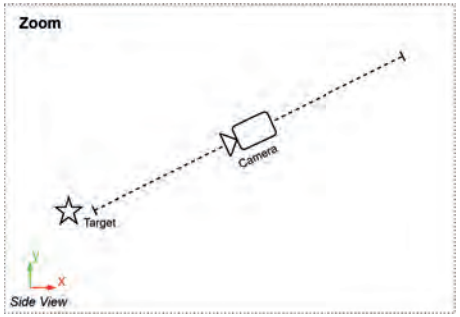
La caméra possède un comportement similaire à celui d'un objet en orbite autour d'un point vers lequel elle regarde. Elle a la possibilité de tourner librement autour du pivot sur l'axe Y, mais est restreinte sur les axes X et Z, pouvant pivoter de 35 à 75° empêchant alors le joueur de regarder sous l'environnement de jeu.



La caméra est faite de telle façon que l'environnement soit au centre de l'attention, elle gravite autour et permet au joueur d'inspecter chaque recoin de l'environnement grâce à une fonction de zoom/dézoom.

Si le joueur clique sur un personnage, la caméra va s'accrocher au personnage et suivra ses déplacements. La caméra se détache si jamais le joueur la déplace alors qu'elle est focus sur un personnage.





Character

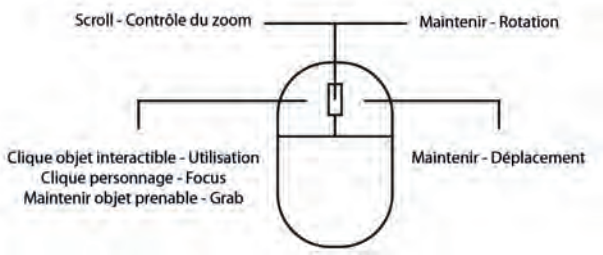
Le joueur contrôle le curseur, c'est sa représentation en jeu, il n'incarne pas de personnage. Lors de ses différentes actions, le curseur sera amené à changer pour apporter un feedback à l'action réalisée. Le curseur peut prendre cinq formes différentes correspondant aux différentes actions réalisables par le joueur :

Déplacer la caméra, faire pivoter la caméra, survoler un objet interactif, attraper un objet interactif et enfin l'état initial.

Comme dit dans la partie caméra, le joueur à la possibilité de cliquer sur un personnage pour focus la caméra dessus, elle suivra alors ses mouvements dans l'environnement.

Base State | Move Camera | Rotate Camera | Over Object | Grab

Controller



Clique - Changement de focus

Maintenir - Hightlight sur les objets interactible

Noyau jouet et tension ludique

Noyau Jouet

Le Noyau de notre projet est une carte dans un état métastable dans laquelle des personnages non joueurs font des suites d'actions.

Le joueur peut alors interagir avec l'environnement et les objets utilisables par les NPC pour essayer d'empêcher le bon fonctionnement du système. Le joueur doit comprendre le fonctionnement de l'environnement pour outrepasser la capacité de la carte à revenir à son état métastable.

Outils

Le joueur peut se déplacer au dessus de l'environnement.

Il a la possibilité d'attraper des objets et de les déplacer jusqu'à une certaine distance.

Tendance Système

Le système aura tendance à retourner à son état initial en résorbant petit à petit les différentes actions du joueur.

Tendance joueur

Le joueur cherche à casser cet écosystème et déclencher des catastrophes en son sein.

Tension ludique

Le joueur doit utiliser intelligemment les mécaniques afin d'aller à l'encontre de la tendance système.

Mécaniques de jeu

Mouvement

Déplacer la caméra :

La caméra peut être déplacée sur les axes X et Z. Elle peut se déplacer sur le plan horizontal en utilisant l'input de déplacement (maintien du clic droit tout en déplaçant la souris).

Orienter la caméra :

La caméra possède un comportement similaire à celui d'un objet en orbite autour d'un point. Elle a la possibilité de tourner librement autour du pivot sur l'axe Y mais est restreinte entre les angles 35° et 75° sur les axes X et Z.

Objets

Attraper un objet :

Positionner la souris sur un objet interactif (un outline blanc apparaît si s'en est un). Effectuer un clic gauche et le maintenir pour continuer à tenir l'objet attrapé.

Déplacer un objet :

Pour cela, vous devez d'abord attraper un objet, ensuite le déplacez avec la souris en continuant de maintenir le clic. Relâcher le clic dépose l'objet là où se trouve la souris.

Si la souris est déplacée trop loin de la position initiale (point depuis lequel l'objet fût attrapé) alors l'objet ne sera pas déplacé si le clic est relâché.

Mécaniques des Agents

L'environnement est rempli de plusieurs agents, tous les agents fonctionnent sur un principe de besoins. Chaque agent possède des jauges représentant leurs différents besoins, ces jauges sont appelées Need Bar et ont tendance à se vider avec le temps.

Need Bars :

Les Need Bar sont des jauges qui se vident donc au cours du temps. Pour pouvoir remplir une de ses need bar, un agent à besoin d'effectuer une Action auprès d'un Stand.

Stands :

Un Stand est un objet présent dans l'environnement vers lequel les agents vont venir effectuer une action pour faire augmenter l'une de leurs need bar.

Actions :

Une Action permet aux agents de faire augmenter leurs jauges, elle dure un certain temps et elles peuvent nécessiter un objet et / ou peuvent créer un nouvel objet.

Boucles de Prédiction

Situation de jeu 1

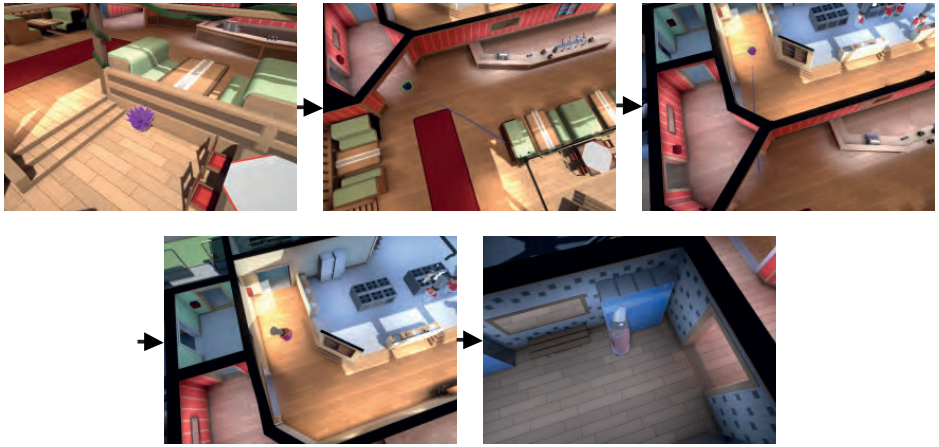
Prédiction : Possède un objet violet, le joueur suppose que s'il le donne à un cuisinier, il modifiera son comportement.

Décision : Le joueur décide de déplacer l'objet pour le mettre sur le chemin du cuisinier.

Action : Le joueur utilise la mécanique de déplacement des objets pour le mettre devant le cuisinier.

Régulation : Le cuisinier s'arrête, ramasse l'objet puis se dirige vers les casiers.

Apprentissage : Lorsqu'un cuisinier ramasse l'objet violet, il va immédiatement le déposer dans un casier.



Situation de jeu 2

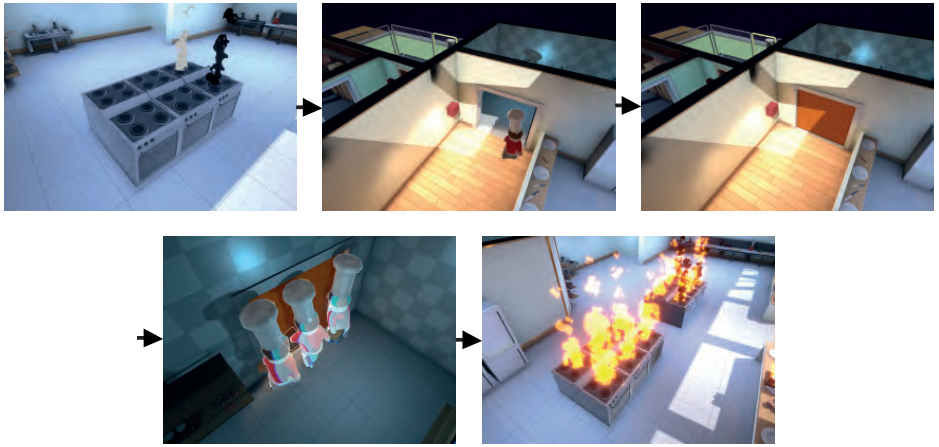
Prédiction : La fumée du four passe progressivement du gris vers le noir, le joueur suppose qu'au bout d'un certain temps le four peut prendre feu.

Décision : Le joueur décide de distraire les cuisiniers suffisamment longtemps pour mettre le feu.

Action : Le joueur ferme la porte de la chambre froide lorsque tous les cuisiniers sont à l'intérieur.

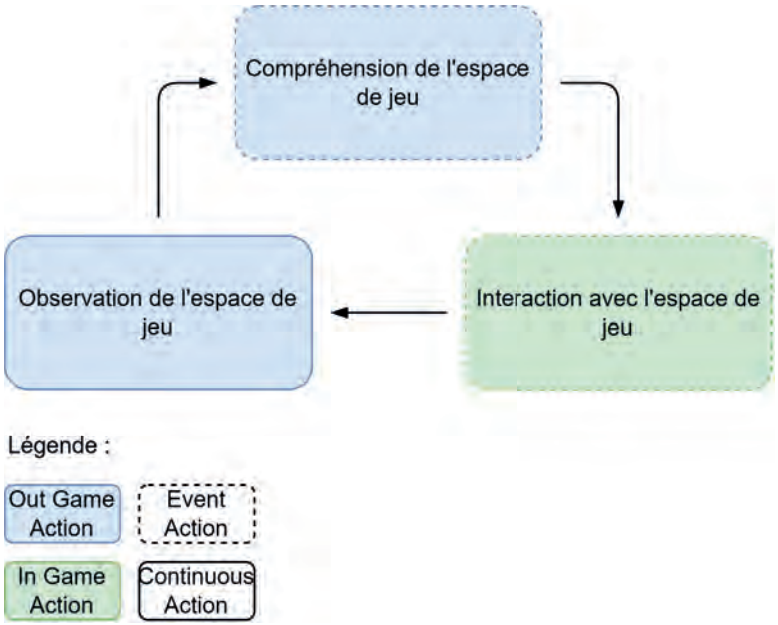
Régulation : Les cuisiniers se retrouvent bloqués à l'intérieur de la chambre froide. Le four prend feu au bout de 20 secondes.

Apprentissage : Le four prend feu, le feu s'étend aux autres meubles aux alentour, le seul moyen d'ouvrir la porte de la chambre froide est à l'extérieur de celle-ci.



Boucles de Gameplay

Guardiola Première Itération (courte)



Guardiola Seconde Itération (longue)

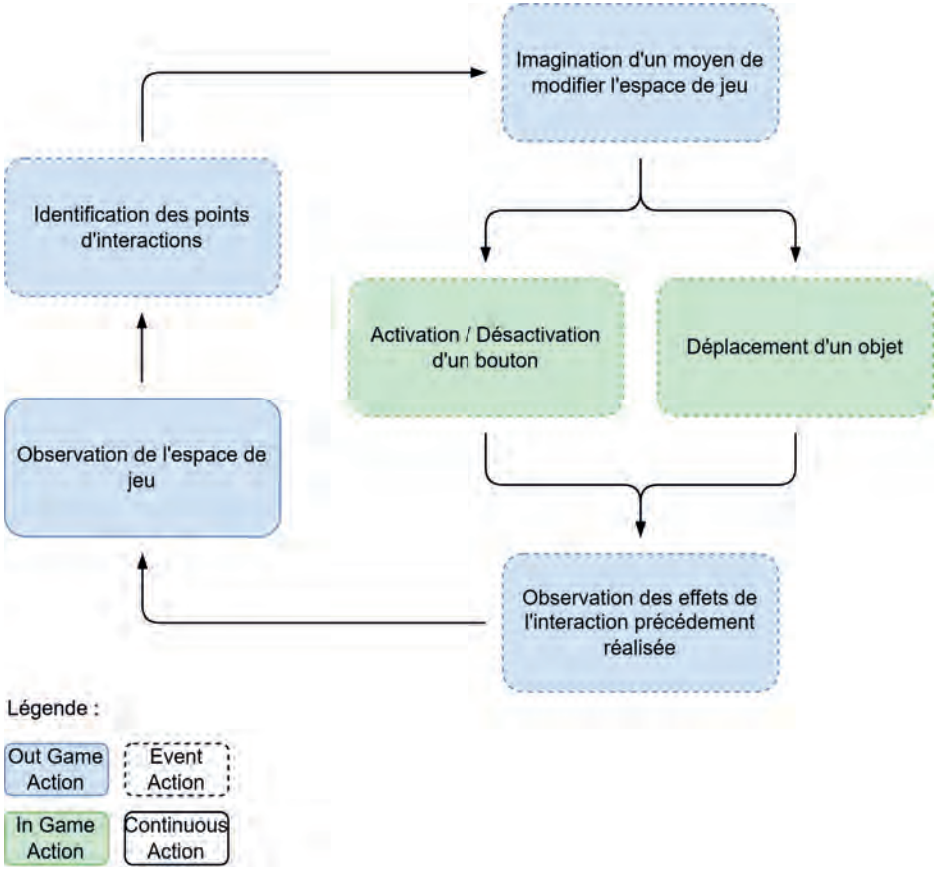
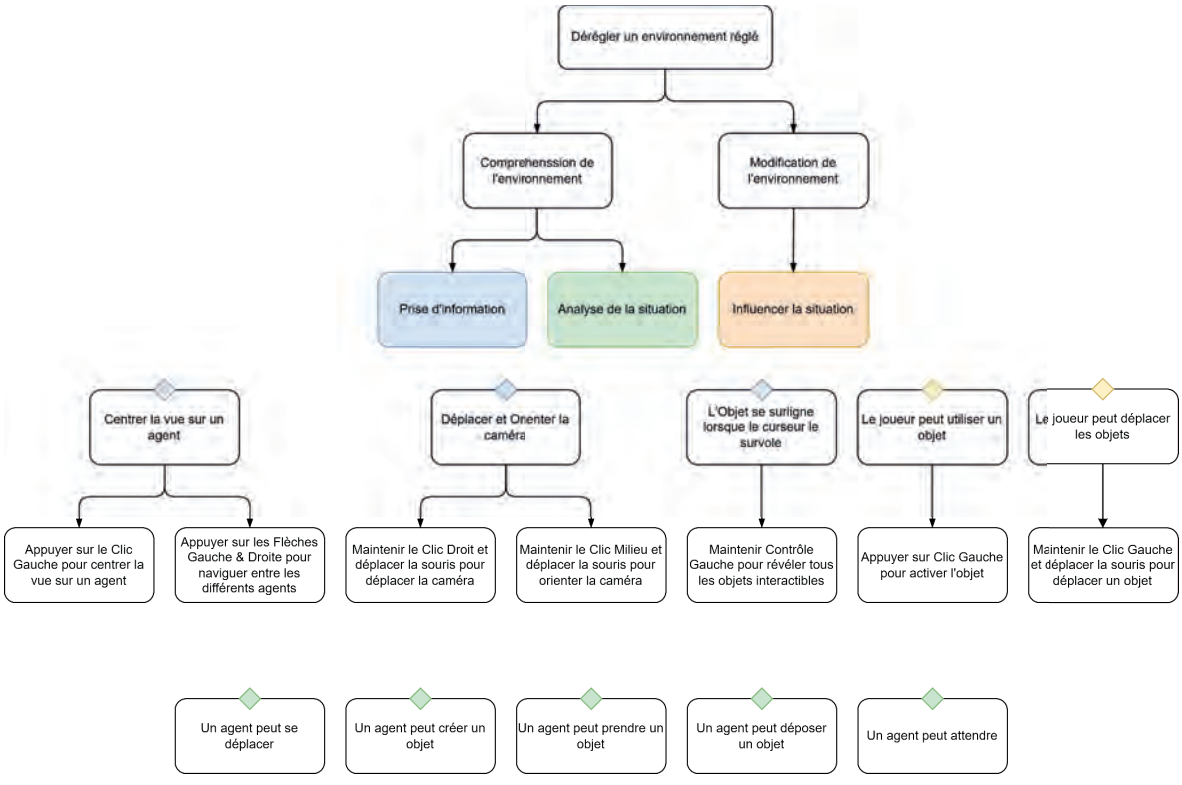


Diagramme de Ventrice



Player Motivation

Sauces

Boucles OCR

Tableau Sign Action Feedback

Boucles OCR

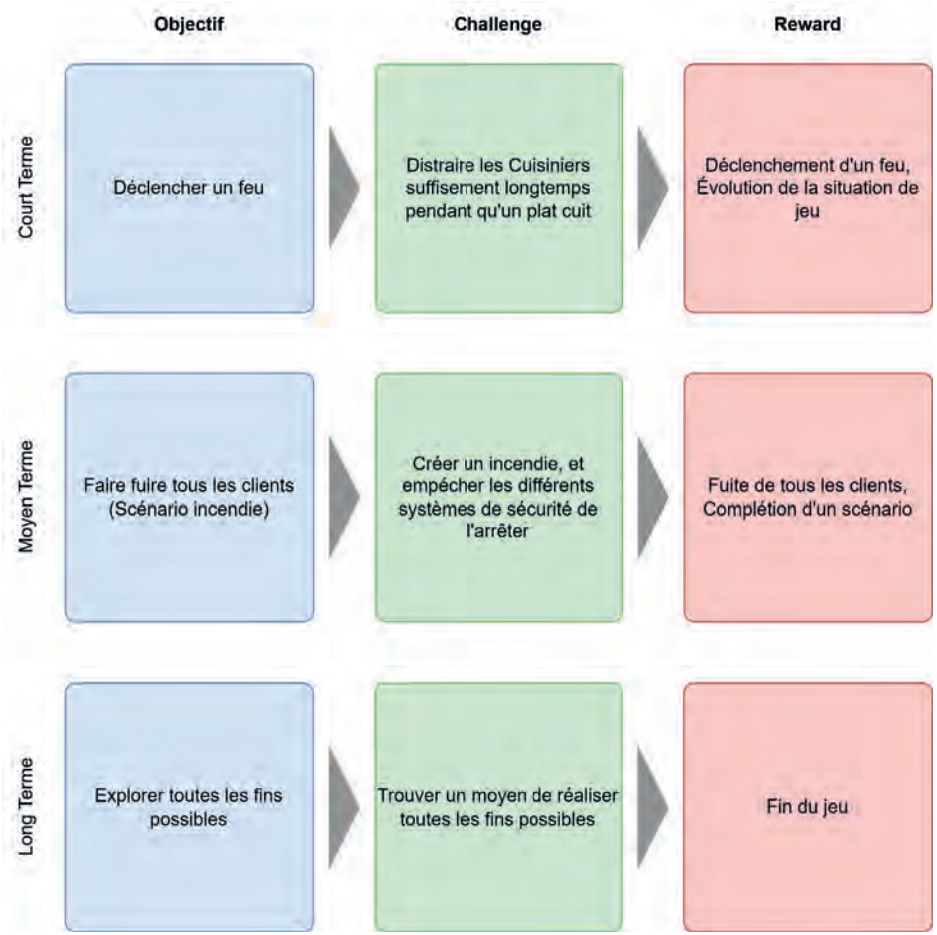


Tableau S.A.F.

	Sign	Action	Feedback
Controlles de la caméra			
Déplacer la caméra	N.A	Maintenir Clic Droit puis déplacer la souris	Déplacement de la caméra
Orienter la caméra	N.A	Maintenir Clic Molette puis déplacer la souris	Rotation de la caméra
Zoomer / Dézoomer la caméra	N.A	Faire rouler la molette de la souris	Zoom de la caméra
Suivre un agent	Agent présent dans le champs de vision du joueur	Clic Gauche sur l'agent	Caméra centrée sur l'agent, début du suivi continu du même agent
Arrêt du suivi d'un agent	Caméra entrain de suivre un agent	Maintenir Clic Droit puis déplacer la souris	Déplacement de la caméra, arrêt du suivi de l'agent
Agent suivant	Caméra entrain de suivre un agent	Pression simple sur Flèche Droite	Changement de l'agent suivi
Agent précédent	Caméra entrain de suivre un agent	Pression simple sur Flèche Gauche	Changement de l'agent suivi
Interaction avec l'environnement			
Activer / Désactiver un bouton	Bouton présent dans le champs de vision du joueur	Pression simple sur Clic Gauche	Activation / Désactivation de l'objet lié au bouton
Déplacer un objet	Objet déplacable présent dans le champs de vision du joueur	Maintenir Clic Gauche puis déplacer la souris	Déplacement de l'objet

Level Design

Poissons

Intentions

Itérations

Map Finale

Logique

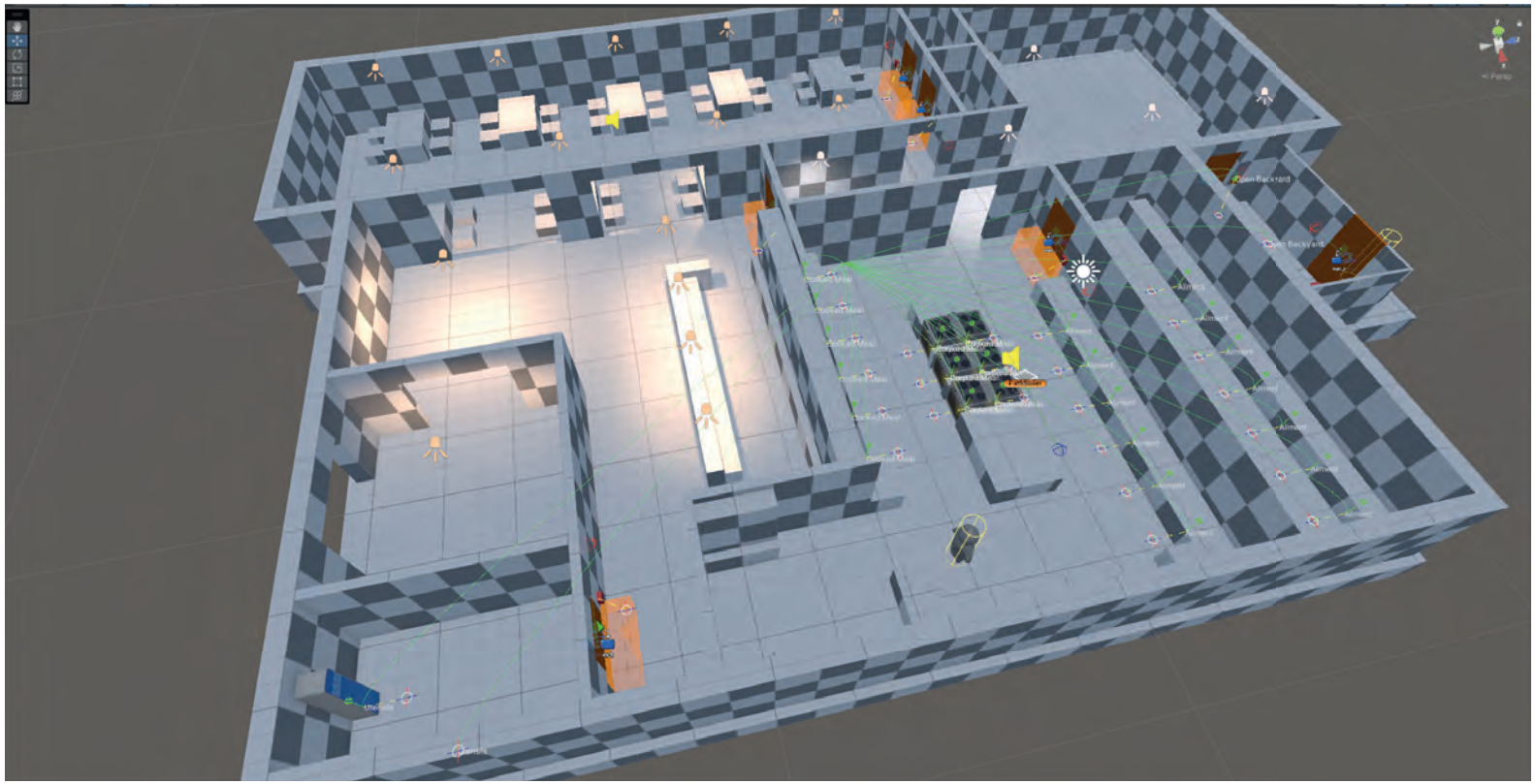
Intentions

Lors de la réalisation de notre jeu nous sommes passés par plusieurs phases concernant le level design. Nous avions des doutes et nous ne savions pas trop comment nous y prendre mais au fil des itérations nous avons su trouver un design de niveau adapté à la morphologie de notre gameplay.

Dans ce jeu nous souhaitons par le level design proposer une diversité de zones dans lesquelles les personnages peuvent réaliser diverses actions.

Nous voulons avec la disposition des différentes portes dans l'environnement permettre au joueur d'entraver les personnages dans leurs actions pour essayer de faire sortir l'environnement de son état métastable.

Nous proposons une diversité d'environnements malgré un espace de jeu global clos.



Itérations

Pour notre première itération, notre objectif était de trouver des environnements où plusieurs personnages pourraient interagir et avoir des comportements différents. Nos premières idées étaient un hôtel et un centre commercial américain.

Bien qu'elle correspondait à nos critères d'un niveau gameplay et esthétique, le problème de ces endroits était le temps de production qui aurait été bien trop conséquent.

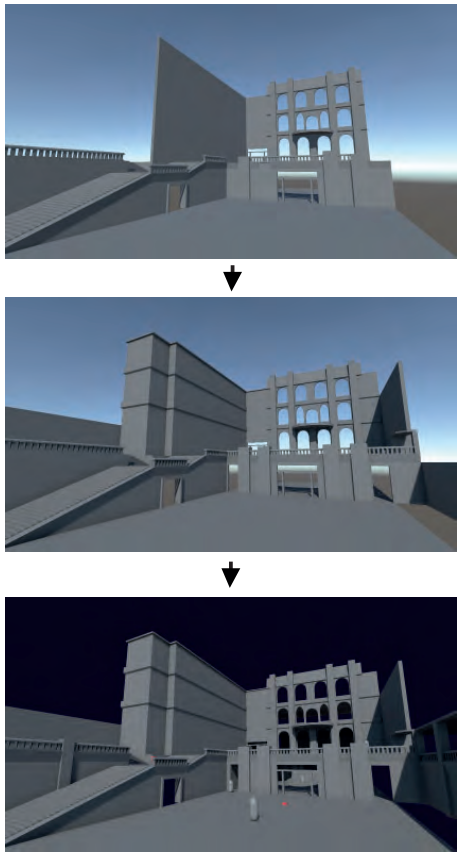
Une première map fût réalisée dans l'esprit d'un palace afin de nous rendre compte de la taille de l'environnement ainsi que des diverses contraintes qu'elle allait amener. Suite à cela nous nous sommes rendus compte que c'était bien trop grand et que pour tester il nous faudrait d'abord un environnement plus restreint.

Nous avons donc décidé de prendre un environnement présent dans les précédentes idées d'espaces de jeux tout en étant plus petit : un restaurant.

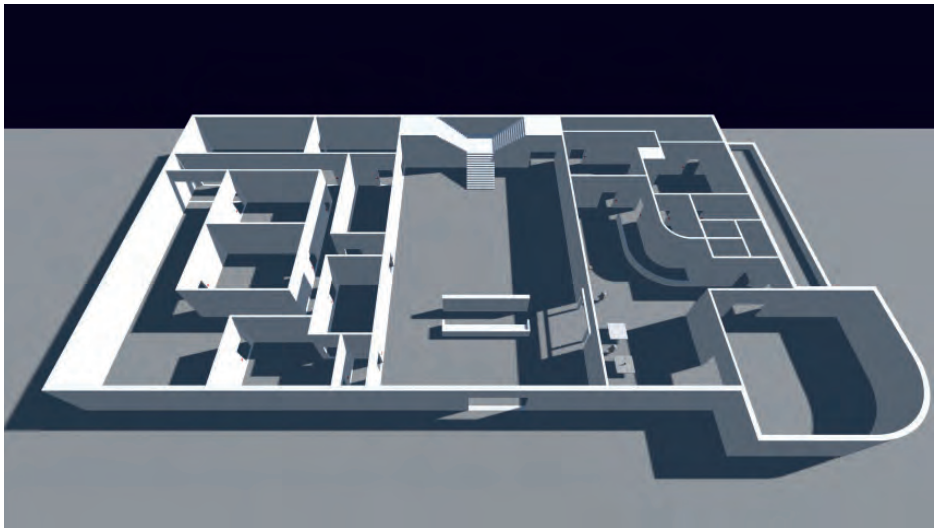
Ensuite, nous avons créé un premier layout avec avec une cuisine, une chambre froide, un bar, une salle et des toilettes. Et pour les personnages, nous avons créé un chef cuisinier, un commis de cuisine, un barman et un homme à tout faire. Chacun avait des envies et des routines différentes.

Le but était de séparer clairement les différentes zones que ça soit visuellement et du point de vue du gameplay. Il fallait que le joueur identifie clairement le rôle de chaque personnage dans chaque pièce. Cet environnement nous a également permis d'effectuer un premier jet pour la direction artistique, pouvant nous donner une idée de ce à quoi pourrait ressembler l'environnement par la suite.

Le problème de ce niveau était que le joueur n'avait pas le temps de faire des actions avant que les personnages ne résolvent les problèmes. Il fallait donc créer une deuxième map plus grande.



Voici trois images en ordre chonologique de création de la première map intégrée dans notre prototype, comme on le voit ici, la capsule parait vraiment petite comparé à l'environnement. Ensuite, suivant nos choix de caméra, cet environnement n'aurait de toute façon pas pu être utilisé car trop axé sur la verticalité.



Test de map sur un seul niveau

Map de test restaurant n°1

Itérations

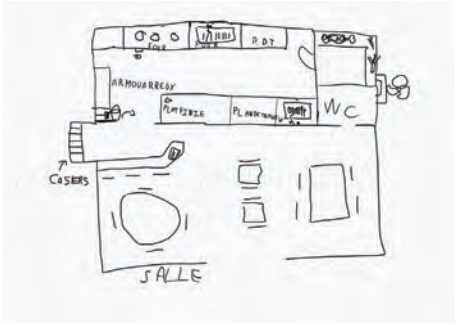
La m_95_Restaurant

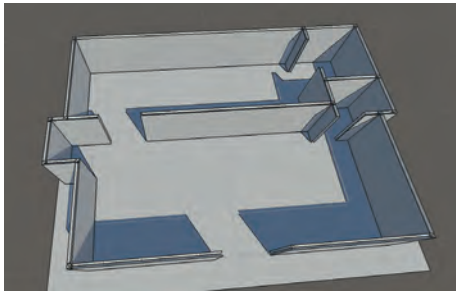
Pour créer notre première map de restaurant nous sommes passés par différentes étapes clés telles que un croquis, un premier blockout, une version meublée puis enfin une version finale. Chacune de ces phases nous a permis de nous rendre compte que nous semblions avancer dans la bonne direction. Ce choix de commencer avec un environnement plus petit s'est avéré très constructif et nous a permis d'itérer rapidement alors que le projet était entrain de stagner.

Voici donc ci-contre le croquis que nous avons réalisé et duquel nous nous basé pour réaliser le premier blocking situé juste en dessous.

Suite à ce premier blocking sur Unity nous avons essayé de remplir la map avec quelques objets afin de contextualiser les pièces entre elles. Cela nous a permis de bien séparer les zones et de commencer à envisager un environnement avec différents archétypes de NPC, des cuisiniers, des barmans/serveurs, un nettoyeur et des clients.

Cette map est un véritable pilier pour notre projet et il est certain que si nous ne l'avions pas fait nous n'en serions pas là aujourd'hui.





La m_32 et la m_33

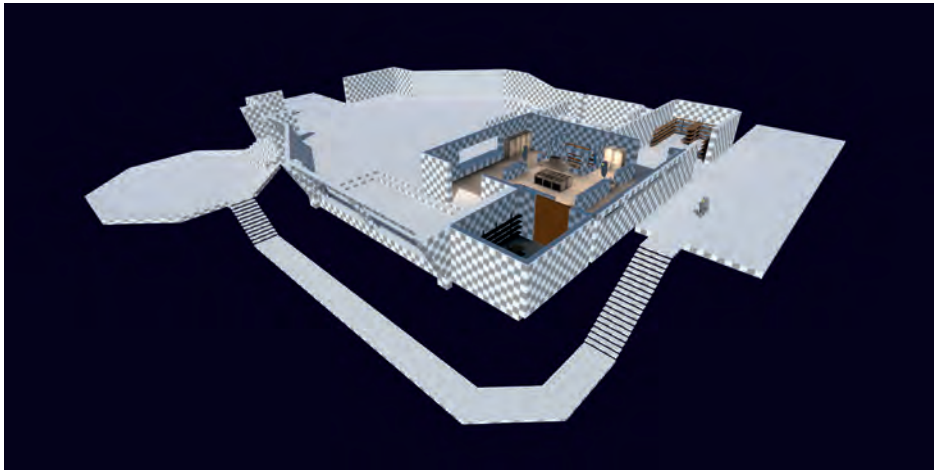
À partir de là, nous avons créé deux versions du restaurant en parallèle.

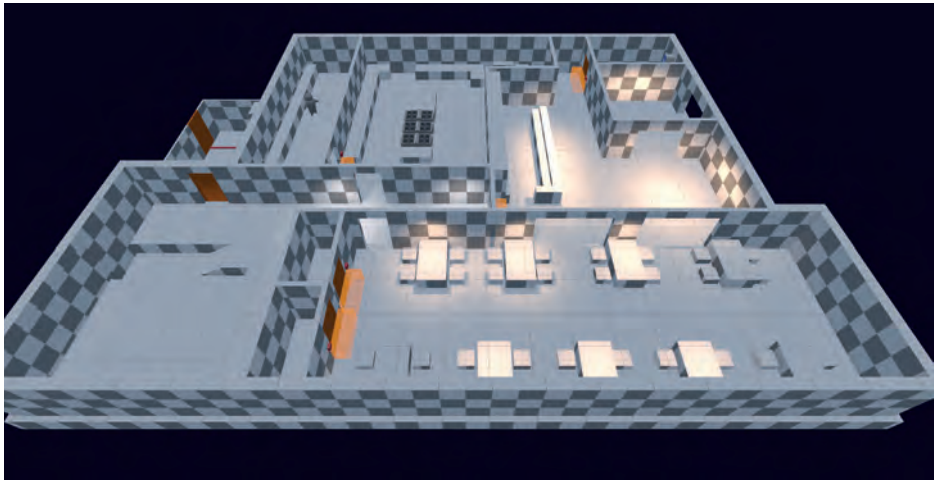
La première était centrée sur la verticalité et l'agencement de la cuisine. Le but était de voir les effets de cette verticalité sur l'esthétique et le gameplay. Nous en avons conclu qu'elle permettait d'ajouter des volumes intéressants mettant en avant certaines zones mais qu'il fallait faire attention avec certaines hauteurs afin de ne pas gêner les mouvements de la caméra.

L'agencement de la cuisine quant à lui est fait de sorte à ce que les personnages puissent tourner autour des fours, et ainsi avoir plus de liberté de mouvement.

La seconde ajoutait une nouvelle salle, l'entrepôt. Elle sert à entreposer des outils et à recevoir des marchandises via un camion qui arrive périodiquement pour réapprovisionner la chambre froide.

La première image est donc la m32 et la seconde la m33.





Map Finale

La m_40_Solstice

Pour finir, notre dernière itération est un mélange et un recalibrage de toutes les idées vues précédemment pour créer un niveau à la fois grand, avec des zones intéressantes, un peu de verticalité pour aider à la séparation de celle-ci et où les personnages peuvent se déplacer facilement.

Comme pour les cartes précédentes nous avons répété notre process créatif en passant par un croquis puis en réalisant un blockout.

Vous trouverez ci-contre le croquis réalisé en groupe lors d'une de nos réunions. De ce croquis découlera le premier blockout de notre map.

C'est avec cette map que nous avons pu mettre en application notre workflow sur blender pour recréer le layout initialement fait sur Unity.

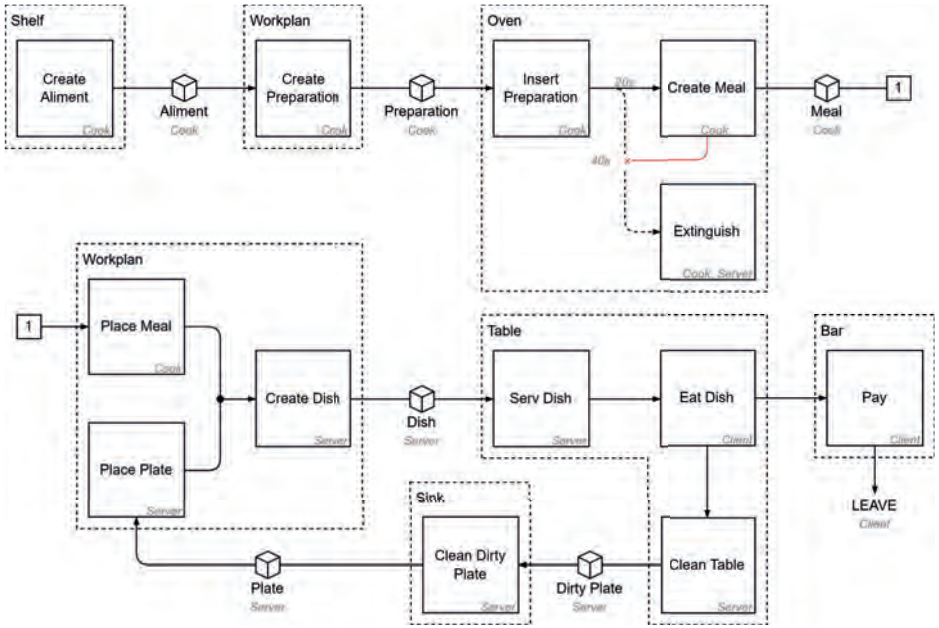
Cela a permis à nos artistes de directement modifier l'environnement pour contribuer à la création de la direction artistique et des assets qui en découlent.

Logique

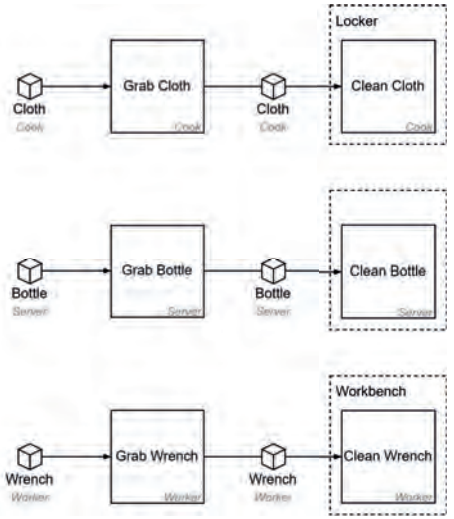
Organisation

Dans le level design de notre jeu nous avons eu besoin d'inclure une partie plus tournée technique, nous permettant de mettre en place les situations de jeu imaginées par les game designers. Pour cela nous avons créé trois outils plaçables et configurables, les items, les stands ainsi que les advertisers.

Ces trois outils nous permettent de créer toutes sortes d'actions que pourront réaliser les npcs comme par exemple créer un aliment puis aller le cuisiner ou encore aller réparer un générateur. Ci-dessous se trouve un schéma décrivant une chaine d'action complète.



Légende :



Pour permettre une bonne réaction du système face aux actions du joueurs, il nous fallait également créer des chaînes d'actions plus réduites, pouvant se répéter. Ces chaînes, ces outils, permettent au jeu d'être plus maléable, moins rigide vis à vis des actions du joueurs. Ces chaînes surpassent en terme de priorité les chaînes de plus grande taille, ce qui donne aux joueurs un outil supplémentaire pour dérégler l'environnement.

Charte Graphique

• Accompagnements •

Univers & Références

Objets

Environnement

Personnages et UI

VFX

IA

Direction Artistique

Univers

À la suite nombreux changements notamment liés aux retours que l'on a reçu lors de notre examen de mi-parcours, nous avons remanié et ce, à plusieurs reprises le game design de notre jeu. Nous avons voulu repartir de zéro et c'est donc pourquoi la direction n'y a pas échappé.

Notre jeu, aujourd'hui permet aux joueurs de jouer avec un petit écosystème, le but des joueurs étant de semer le plus de chaos possible, à tel point que l'écosystème ne puisse plus résorber les actions du joueur par lui-même. Ces mécaniques évoquent quelque chose d'amusant, de défoulant c'est pourquoi nous avons décidé de diriger la direction artistique sur un axe plutôt humoristique, tendant à faire rire les joueurs avec des situations loufoques qu'ils pourront eux même engendrer.

Qui dit humour, amusant, défoulant dit haut en couleur. Nous avons décidé dans ce jeu d'apporter beaucoup de couleurs pour créer une ambiance chaleureuse, bon enfant, avec des textures illustrées, nous rapprochant des cartoons qui nous inspirent par leur coté très maladroit (au niveau des personnages) et leur façon de créer des environnements à la fois lisibles et très colorés.

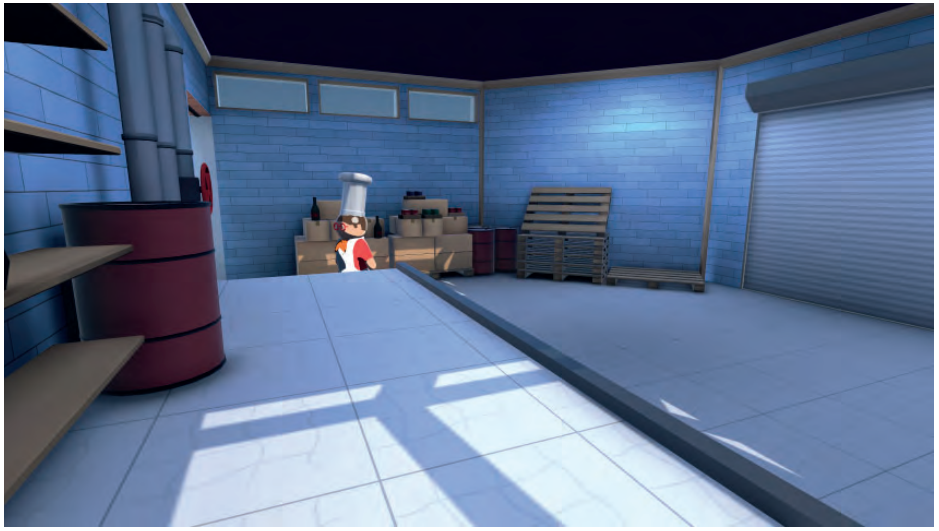
Intentions

- La chose la plus importante dans un environnement chargé et coloré c'est sans aucun doute la lisibilité. Nous voulons de part nos designs permettre aux joueurs une bonne lisibilité lors de leurs sessions de jeu. Cela passe par plusieurs procédés tels que l'utilisation de shaders pour contraster sur des effets brillants mais aussi par la colorimétrie et les lumières afin de contraster entre les personnages, les objets et l'environnement.

-Nous voulons que la direction desserve le gameplay humoristique du jeu en accentuant les situations loufoques grâces à des animations, des effets visuels et sonores inspirés du monde du cartoon.

-Il est de notre souhait de diversifier l'expérience visuelle du joueur grâce à une utilisation astucieuse des couleurs permettant de bien séparer les différentes pièces de l'environnement. Pour cela nous avons réalisé un traitement au niveau des textures sur l'entièreté de l'environnement, le séparant en trois grandes zones : la salle de restaurant (boisé, couleurs chaudes), la cuisine (carrelage couleurs plus claires, moins saturées) et enfin les locaux techniques (murs de briques et sol fait de dalles).

-Nous portons une attention particulière sur les feedbacks visuels en jeu. En effet, de par la grandeur de la carte, le nombre de personnages et d'actions dans l'environnement, il est très important pour nous d'arriver à correctement signifier l'état du système au joueur. Cela passe par la création d'animations courtes et facilement compréhensibles, par le fait que notre environnement soit inspiré d'objets, de structures du monde réel ainsi que par la réalisation d'effets visuels puissants par leur façon de contraster avec l'environnement.

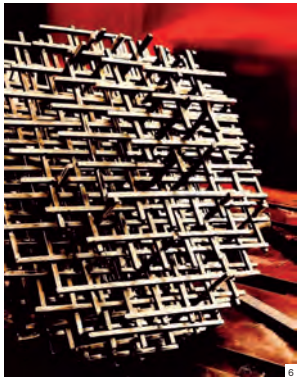


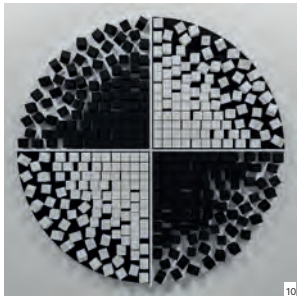
Moodboard

Recherches lignes et contours

1. *RETROSPECTSilkscreen* **Keith Haring**
2. *Sculpture* **Eric Martineau**
3. *Dr Stone Figure Repaint* **m_a_man**
4. *Ah! Storm clouds rushed from the channel causts* **Patrick Caulfield**
5. *Zeta Gundam Head Repaint* **Lyk Repaint**

6. 7. *Tumbleweed, Fish Egg Maquette* **Philip Vaughan**
8. *La Ligne* **Franz Alias**
9. **Dessin animé la Linea**





10. *Interaction*
Andrij Savchuk

11. 12. *Flight of imagination, Walking the line*
Paul Brouns

13. *Vitraux Abbaye Sainte-Foy de Conques*
Pierre Soulages

14. 15. *Œuvres non baptisées*
Pierre Soulages

Références Commentées

Lignes et Contours

Nous avons décidé d'effectuer un travail de recherche sur les lignes et les contours, notamment dans des domaines extérieurs au jeu vidéo. Mais pourquoi ce travail sur la ligne et les contours ?

Pour la création de notre shader de contours il nous était nécessaire de bien comprendre comment signifier correctement ces deux notions artistiques élémentaires. Naturellement les œuvres de *Keith Haring* se sont montrées particulièrement inspirantes, sa façon d'exprimer le mouvement et les contrastes est très intéressante. Un autre artiste qui nous a marqué par son travail du contour et cette fois ci également de la couleur, *Patrick Caulfield*. Ce pont entre les deux artistes est palpable, en termes de couleurs, de trait, ce qui nous a attiré notamment chez *Caulfield* c'est la profondeur et cet effet de relief qu'il crée dans ses œuvres, ce qui en fait une bonne source d'inspiration pour nous qui réalisons en 3D.

Entre 2 et 3 dimensions

Dans un autre registre, il existe certains maquettistes/artistes qui pratiquent du "Repaint" sur des figurines ou autres objets pour leur ajouter un style. Ici nous mettons l'accent sur les "Repaint" qui façonnent l'objet de telle sorte qu'il semble provenir d'un dessin animé ou d'une bande dessinée. Ce que nous voulons c'est toucher du doigt dans notre design la frontière entre la 2D et la 3D, c'est pourquoi les œuvres 2D qui se veulent 3D et les Objets 3D qui se veulent 2D nous intéressent tout particulièrement, fréquentant chacun d'un côté différent cette fameuse frontière.

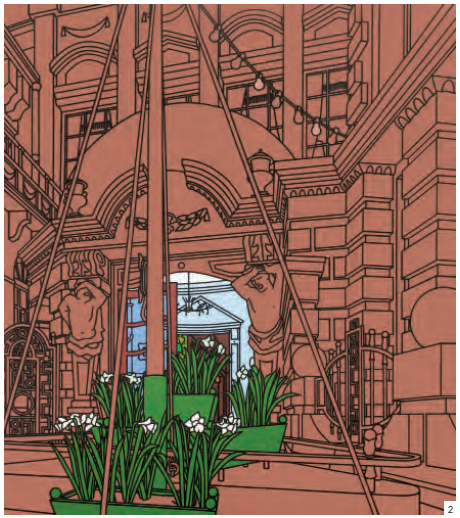
Nous souhaitons créer un shader qui puisse faire en sorte de rendre facile l'interpretation visuelle des objets dans notre jeu, pour cela nous allons jouer avec les couleurs et les lignes sur les formes / arêtes de nos différents objets.

Nous avons vite compris que le travail de la ligne était surtout une histoire de contraste. Un contraste souvent net et puissant qui sépare brusquement un sujet pour y définir une forme. Un artiste qui nous a particulièrement inspiré sur ce sujet se trouve être *Pierre Soulages*. Cet artiste arrive à créer du contraste sur ses peintures tout en utilisant uniquement de la peinture noire.

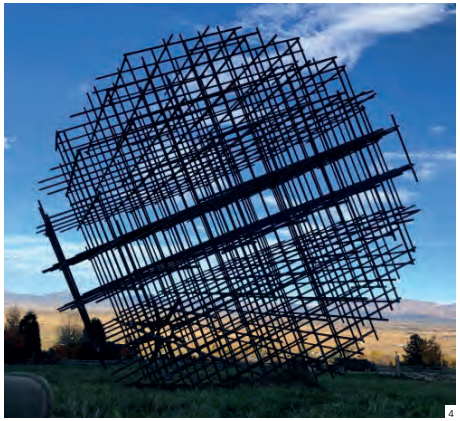
Nous avons donc compris grâce à lui que ça n'était donc pas seulement une histoire de couleur ou de teinte mais également une histoire de lumière et de texture. Quand on parle de lumière et de relief, les oeuvres de *Philip Vaughan* se sont révélées intéressantes, ses sculptures, à la fois denses en complexes semblent être des oeuvres tout à fait différentes sous chaque angle, le travail de la lumière sur le métal permet de façonner l'expérience de la même manière qu'elle le fait sur les oeuvres de *Pierre Soulages*.

Ces recherches nous ont permis d'entrevoir de multiples possibilités pour façonner à notre tour les objets de notre projet notamment pour la création de notre shader d'outline pour le survol des objets interactifs. Cet outil nous permet de faciliter la compréhension en jeu et de générer le feedback permettant au joueur de comprendre qu'il peut se passer quelque chose avec l'objet.

1


2


3


4

1. *Dogs* **Keith Haring**
2. 3. *Forecourt, Foyer* **Patrick Caulfield**
4. *Atom – Rebar Sphere* **Philip Vaughan**
5. **Pierre Soulages**


5

Références Commentées

Petites silhouettes

Nous voulons avec notre direction artistique rendre un aspect bon enfant, haut en couleurs, rigolo. Il nous fallait des références pour nos personnages pour les rendre petits, maladroits, que ça soit au niveau de la forme du corps ou au niveau des animations.

La première oeuvre présente ci-contre est une oeuvre d'*Andrea Benetti* met en scene plusieurs petits personnages colorés. Plusieurs caractéristiques nous intéressent dans cette oeuvre, elle est colorée, possède de petits personnages semblant comme attirés vers un point de convergence au centre. Cette oeuvre nous fait penser à l'aspect incontrôlable de notre jeu, s'emballant dès lors que le joueur commence à mettre un peu de bazar dans l'environnement.

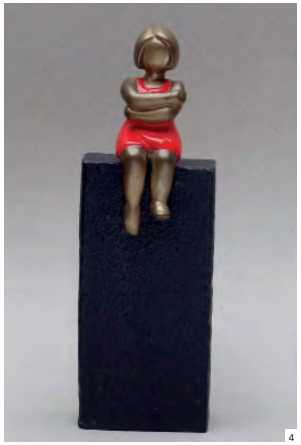
On peut retrouver l'aspect enfantin, innocent et bedonnant de nos personnages dans l'oeuvre de *David Law* . On apprécie le contraste entre les figurines et le monde réel dans ses oeuvres. La façon d'exprimer les échelles de tailles ici nous intéresse aussi, là où nous devons rendre les actions de nos personnages amusantes et maladroites, quoi de mieux que de petites proportions inspirées de celles d'un enfant.

La première, la dieuxième, la quatrième ainsi que la cinquième oeuvre ont toutes un point commun qui pour nous a son importance : la silhouette. Pour ne pas nous tromper dans notre design de personnage il est important pour nous de nous inspirer de ces oeuvres représentant des personnages aux menues proportions. On remarque que les personnages ont des mains et pieds simplifiés, parfois pas de visage mais arrivent quand même à signifier certaines expressions comme c'est le cas dans l'oeuvre de *Marie Ange Pol*.

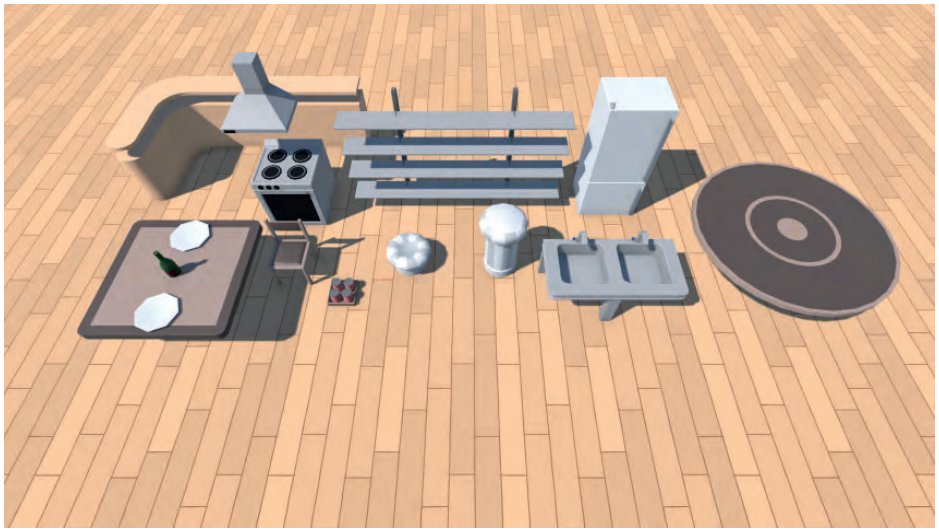
L'oeuvre de *Nurit David* faitr le pont avec le précédent moodboard, l'effet de contraste ici est très important, accentué par la présence d'un contour autour de ce que l'on immaginera être deux petites têtes. Nous nous sommes posé la question quant à la nécéssité de donner des yeux ou un visage à nos personnages mais du fait de la distance entre les personnages et la caméra mais aussi dans un soucis de densité nous avons choisi de nous inspirer de la *Boudeuse sur mur*.

Le theme de la cuisine et de la restauration sont au centre de notre jeu, c'est pourquoi le *Vegetarian Kebab* vient se balader dans nos recherches. L'approche est différente comparé aux autres oeuvres, celle-ci ne nous parle pas spécialement de formes rondes ou de personnages maladroits. Ici le petit cuisinier est positionné devant sa carotte et semble attendre, sûr de lui, devant ce pilier végétal. Cette oeuvre est là pour rappeler qu'il est d'autant plus important de savoir maitriser lorsque l'on produit, design quelque chose. Nos personnages se doivent en effet d'avoir cette étiquette du petit bonhomme maladroit et burlesque mais il est tout aussi élémentaire de savoir correctement jauger afin de ne pas tomber dans l'excès et de nous même devenir des êtres bien maladroits...

1. *Aborigeni* **Andrea Benetti**
2. *Washing Machine* **David Law**
3. *Untitled (Shirt 2)* **Nurit David**
4. *Boudeuse sur mur* **Marie Ange Pol**
5. *Lamppost* **Katherine Lubar**
6. *Vegetarian Kebab* **David Gilliver**



Place Holders et Palette



Pour faciliter la création de la carte et permettre aux designers de mieux imaginer leurs situations de jeu, nous avons créé un grand nombre d'assets place holders.

Nous avons tout d'abord créé un grand nombre d'objets sans thématisation puis nous nous sommes vite concentrés sur un environnement typé "restauration".

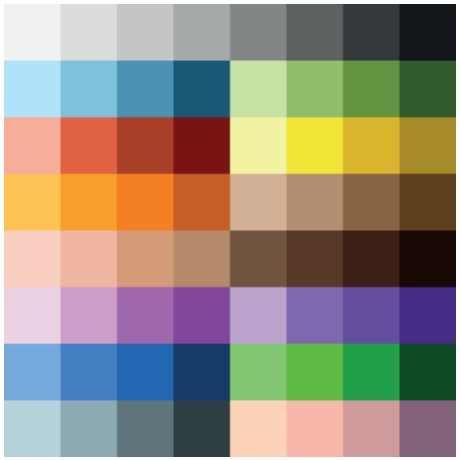
Dès la création de ces place holders, il a fallu nous soucier de la taille des objets vis-à-vis de celle que nous voulions pour les personnages. Un petit test nous a vite fait nous rendre compte que des objets un petit peu trop grands dans un environnement peuplé de personnages bedonnants pouvait rendre les situations fort cocasses et amusantes.

Suivant ce premier batch d'objets, nous avons entamé la réelle production des objets de l'environnement, passant pour un premier temps par la réalisation de versions abouties des objets place holder puis l'ajout d'objets supplémentaires permettant d'augmenter le nombre d'interactions pour le joueur et remplissant par conséquent le reste de la carte.

Une texture unique

Pour nous permettre une production rapide et de qualité nous avons opté pour un workflow passant par une palette de couleurs pour nous objets. Ce procédé nous évite de passer par l'utilisation de logiciels de texturing et nous permet donc de gagner un temps précieux.

Pour la première version de nos objets (placeholders) nous avons utilisé une palette trouvée sur le net puis pour la réalisation des assets finaux nous avons créé notre propre palette, qui de par ses couleurs, ses différents tons se rapproche plus de ce que nous voulons exprimer en termes de couleurs (quelque chose de plus vif, plus joyeux, plus saturé). En passant par une palette nous optimisons par la même occasion le nombre de drawcalls en jeu, ce qui nous permet de gagner en performances et de rendre le jeu plus accessible.



Ci-dessus la palette utilisée pour les place holders, cette palette provient du créateur "Imphenzia" et nous a bien servi lors de nos prototypages d'assets.

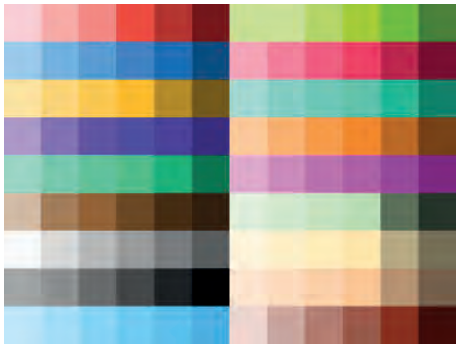
Palette et objets finaux

Remplir l'environnement

Vous pouvez retrouver ci-contre les objets finaux réalisés pour notre jeu. La création de tous ces items fu un défi de taille pour nos deux game artists. Il a fallu pour la carte finale recréer un grand nombre d'objets afin d'ameubler au mieux toutes les zones de l'environnement. Certains objets issus de notre semestre précédent se sont avérés très utiles après quelques petites retouches, collant très bien avec l'esthétique de l'entrepot et de la salle des machines.

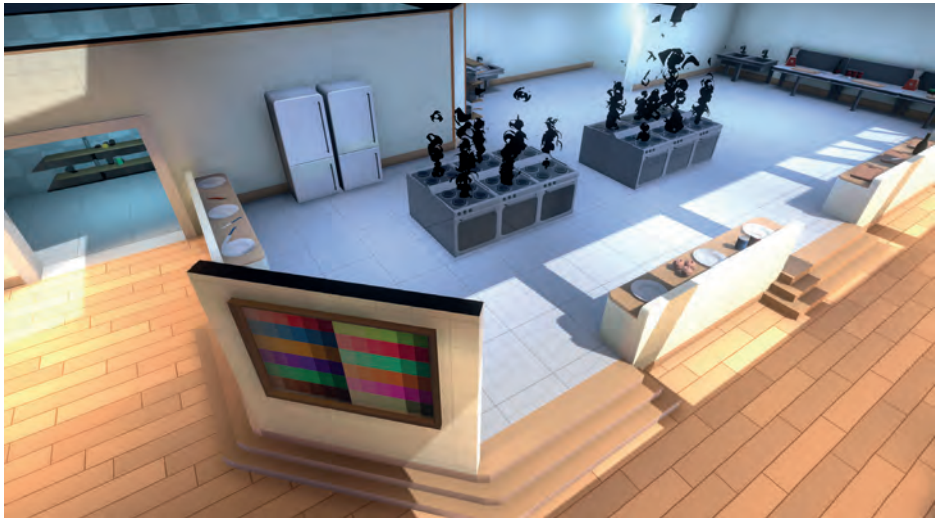
Sur les images sont donc placés les objets en fonction de leur taille, on retrouve ci-dessous les objets les plus gros, ne pouvant être déplacés par les personnages, ils constituent l'esthétique du niveau ainsi qu'une partie du level design, notamment dans la cuisine et dans la salle de restauration.

Ci-contre les objets de plus petite taille dont une majorité peuvent être attrapés par les personnages. Ces plus petits objets nous permettent d'enrichir les mises en situations en les rendant plus vivantes. Les personnages utilisent des couverts, découpent les aliments puis les cuisent dans la poele etc...



Voici donc la palette de couleurs créée par nos soins. Comparativement à la première nous avons ajouté des gammes de couleurs supplémentaites et avons réhaussé la saturation afin d'obtenir des couleurs plus vives et marquantes. Il est important que nos objets soient capables de dénoter de l'environnement afin que le joueur puisse plus facilement les discerner dans la masse d'informations que représente l'espace de jeu.

Environnement



Comme expliqué dans la section Level Design vue précédemment nous sommes passés par plusieurs grandes étapes concernant le design de notre environnement de jeu.

En effet, l'environnement fut tout d'abord testé sur un gabarit de petite taille, nous permettant de comprendre ce que nous voulions et de découvrir certains problèmes notamment au niveau des textures. Pour éliminer ces problèmes nous avons décidé d'utiliser pour notre environnement beaucoup de textures dite "World Align".

Ce procédé, en plus de garantir une très bonne qualité de texture, est un très gros gain de temps car il nous dispensé d'unwrapping sur la structure de notre environnement (murs et sols). Nous avons donc créé, dessiné nos textures nous même puis nous les avons réglés grâce à la configuration du shader dans Unity. (Photo texture sans couleur, avec couleur, puis sur l'environnement, frise).

Pour l'environnement nous avons choisi de passer par des couleurs moins saturées comparativement à celles de la palette utilisée pour les objets afin de contraster légèrement entre les deux.

Bien que très pratique, le "World Align" ou "Triplanar" possède quelques contraintes qu'il nous a fallu braver afin d'assurer une belle qualité graphique. En effet, l'environnement possède un grand nombre d'angles situés entre 0 et 90 degrés or le triplanar ne sait gérer que les angles situés sur les axes c'est-à-dire 0/90/180/270°.

Pour contrer ce problème, nous avons appliqué pour ces murs seulement un shader plus basique sur lequel nous avons réglé la texture pour qu'il n'y ait pas de décalage visuel entre les deux textures.

Personnages

Chara-Design

La grande différence entre notre jeu actuel et celui que nous proposions au premier semestre se trouve dans la présence de personnages entièrement visibles à l'écran. Un choix qui nous a permis de nous faire violence et qui a poussé les artistes à apprendre, itérer sur des questions de character design jusque-là complètement inconnues.

Nous avons tout d'abord créé un petit personnage inspiré de ceux pouvant être retrouvés dans des jeux tels que *Gang Beasts* et *Human Fall Flat*. Ce design à la fois petit, bedonnant et maladroit nous a tout de suite plu et nous en avons fait notre personnage place holder.

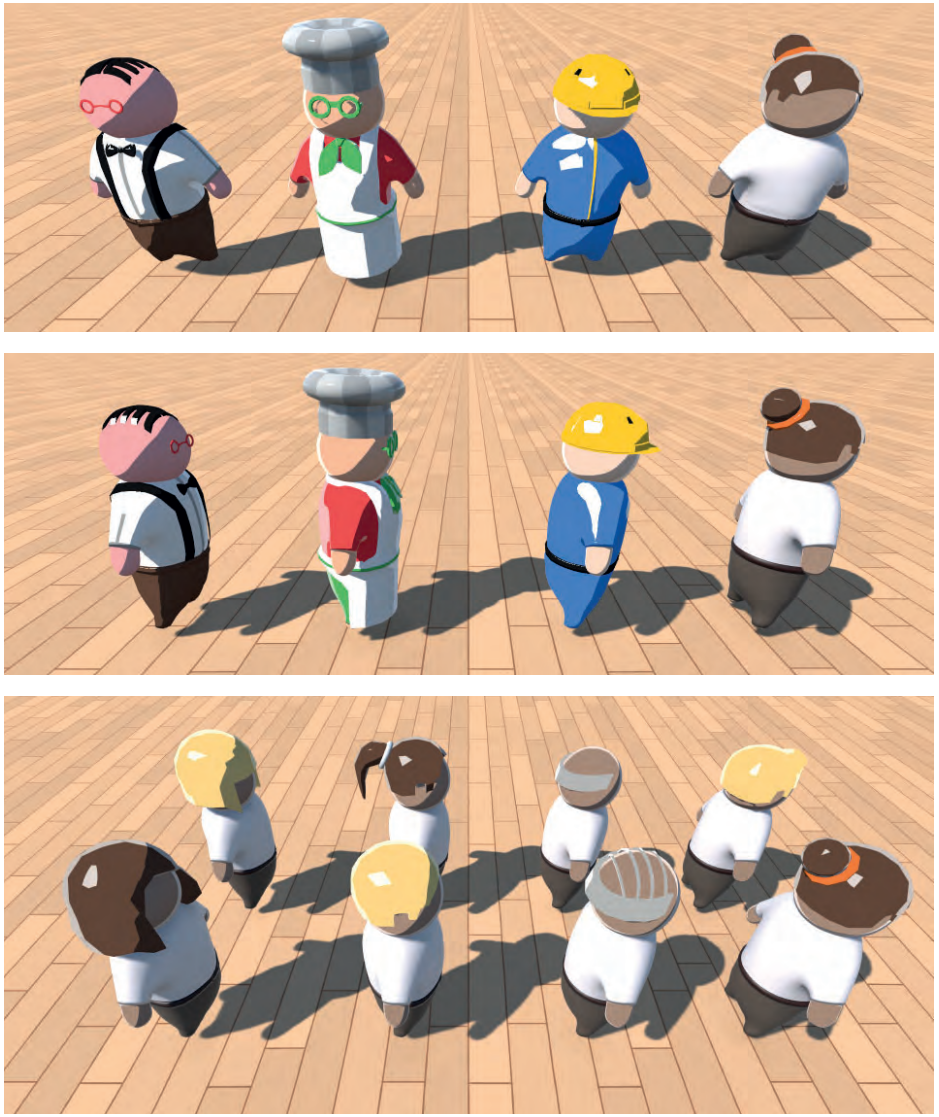
Pour différencier nos personnages en jeu nous avons tout d'abord créé des chapeaux puis nous avons ensuite produit quatre designs se basant sur les proportions de notre placeholder (un cuisinier, un serveur, un technicien et un client). Pour ajouter de la diversité nous avons également créé un total de dix coiffures et couvre chefs, nous permettant de créer des personnages ressemblants mais tous uniques, passant également par une utilisation des couleurs faisant varier celle de leurs habits notamment.

Pour permettre au joueur de bien identifier les personnages nous avons joué sur les clichés et conventions des métiers présents dans notre environnement (le tablier et la toque pour le cuisinier, le casque de protection pour le technicien, le costume et le nœud papillon pour le serveur). Pour encore les différencier de l'environnement nous avons appliqué sur les personnages un shader modifiant l'effet des lumières sur leur corps, les rendant plus brillants en passant par un toon shader.

Comme montré ci-contre il est possible de réaliser une bonne variété de personnages, sachant que toutes les couleurs, que ce soit la peau, les cheveux ainsi que certains éléments des habits peuvent être changés par des matériaux prédéfinis par nos soins dans l'editeur, nous permettant une certaine variété au niveau des visuels des personnages.

Les habits

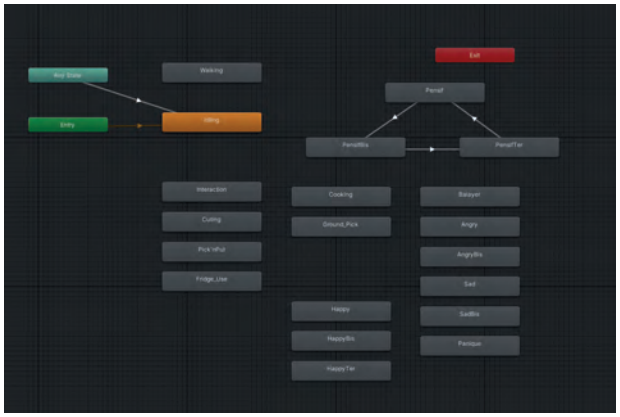
Une fois le personnage créé il a fallu lui faire toute sortes d'habits. Suite à des tests réalisés par l'un des membres du groupe nous avons pu imaginer quatre tenues différentes, un tablier, un bleu de travail, un uniforme de serveur ainsi qu'une tenue décontractée pour les clients. Il était important, pour bien séparer les différents designs de jouer avec les couleurs et les formes, de façon à comprendre du premier coup d'oeil ce qu'est tel ou tel personnage.

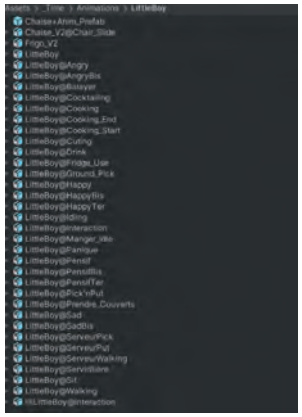


Animation

Tout comme le character design, l'animation nous était presque tout aussi étrangère c'est pourquoi nous avons décider de nous concentrer sur la réalisation de petites animations, peu complexes en premier lieu (attraper un objet, balayer, marcher). Tout au long de la production nous avons gagné de l'assurance sur l'animation et nous en avons réalisé des plus complexes telles que l'animation de panique, celles liées aux émotions ainsi que les animations liées aux clients (s'assoir, manger et boire).

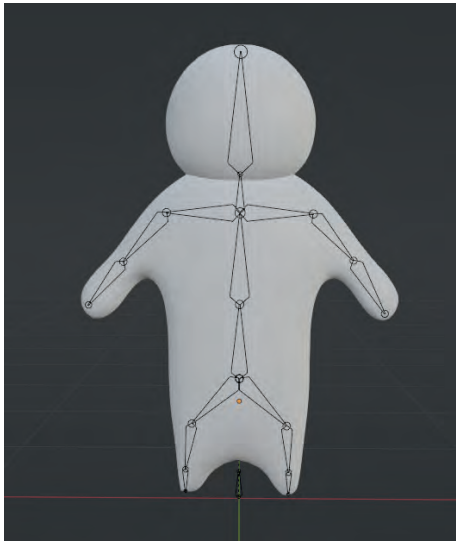
Pour gagner du temps nous avons utilisé un workflow très précis avec des normes de nomenclatures des fichiers permettant à Unity de directement lier l'animation au personnage dans le moteur. Avec ces contraintes d'export nous avons pu produire plus efficacement et de telle manière à ne pas surcharger le projet de choses inutiles en exportant seulement l'animation et l'armature pour chaque nouvelle animation.





Ci-dessus vous pouvez donc retrouver l'animator de notre projet. Ici les animations sont pour la plupart simplement nommées et ajoutées à l'animator car une simple référence à l'animation dans l'outil de programmation des IA nous permet de leur faire jouer l'animation souhaitée en fonction de l'action réalisée. Cet outil est donc très pratique car il nous permet d'itérer rapidement et de pouvoir directement tester en jeu.

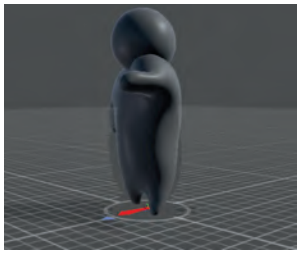
A coté de l'image de l'animator se trouve la liste des animations réalisées pour le projet, comprenant des animations pour des actions physiques réalisées par les personnages telles que l'animation de marche, de balayage ou encore de ramassage d'objet. Nous avons également ajouté des animations représentant des émotions telles que la joie, la colère, le questionnement ou encore la tristesse de façon à rendre nos personnages plus humains, plus attachants.



Voici le rig de notre personnage. Pour nous aider dans l'animation et rendre nos personnage plus maladroits nous avons créé un rig dit "simplifié", ne possédant ni mains, ni visage, ni pieds.

La forme basique de notre personnage nous a permis de rentre les animations plus accessibles pour les membres de l'équipes qui n'y étaient pas habitués, nous permettant alors de produire plus d'animations sur le temps qui nous était donné.

Le fait d'avoir rendu le personnage aussi bedonnant nous a d'abord facilité la tâche puis posé quelques problèmes. En effet de part sa petite taille et sa corpulence il est difficile pour lui de réaliser certaines actions. Cependant, ces problèmes rendent nos animations plus drôles, enfantines, les personnages doivent utiliser certaines techniques qu'utilisent les enfants pour tirer des objets, manger , s'assoir etc...



Sur la première image on voit tout de suite de quoi il s'agit quand on parle du manque d'agilité des personnages. Il doit, pour faire cuire un aliment, lever son bras et son épaul si haut que cela pousse sa tête sur le coté.

Pour récupérer un objet le personnage doit se baisser entièrement à cause de ses touts petits bras, ce qui nous permet, même de loin, de faire comprendre au joueur que le personnage ramasse quelque chose au sol.

La dernière image provient d'une drôle d'animation réalisée par l'un d'entre nous. Cette animation de célébration pourrait être utilisée dans le cas où un personnage arrive à accomplir quelque chose d'important.

User Interface

Souris

Dans notre jeu, la souris est le seul moyen d'interagir avec le monde, nous avons donc décidé de créer un curseur personnalisé qui permettra au joueur d'être plus intégré au monde.

Pour cela, nous avons commencé par déterminer les états de la souris. Elle a donc les états:

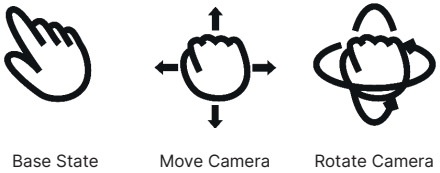
- Base State (état neutre où le joueur ne fait rien et aucune interaction n'est possible)

- Over Object (quand la souris est sûr un objet avec lequel le joueur peut interagir, bouton, personnages, etc)

- Grab (quand un objet est pris par le joueur)

- Move Camera (quand le joueur déplace la caméra)

- Rotate Camera (quand le joueur fait tourner la caméra)



Base State | Move Camera | Rotate Camera

Over Object | Grab

Pour le design des icônes, nous nous sommes inspirés de l'icône de main classique de Windows et nous l'avons rendue plus ronde et douce. Nous avons fait ensuite des déclinaisons en fonction des états que nous voulions, l'icône:

- Default est un mélange entre le pointeur de souris avec un design de main

- Interactable est une main qui pointe un objet (comme quand on appuie sur un bouton)

- Prenable est une icône de main ouverte

- Pris est une main fermer

- Déplacement est une main fermer avec des flèches sur les côtés

- Rotation est une main fermée avec des flèches qui forment des orbites autour de la main.

Personnages

Il nous fallait trouver un moyen permettant aux joueurs de mieux comprendre les prochaines actions des personnages. C'est pourquoi nous avons décidé d'ajouter des bulles apparaîssant lorsqu'un personnage choisit de réaliser une nouvelle action.

Pour ces bulles nous nous inspirons des bulles présentes dans les BD, monde dans lequel ces bulles sont utilisées pour représenter les pensées. Ce choix nous permet de jouer sur les conventions de design et nous assure une bonne compréhension de la part du joueur.

Pour remplir ces bulles nous avons réalisé quelques designs en accord avec les actions que visent les personnages (clé à molette pour réparer, chaise pour l'action de s'assoir, un balais pour l'action de balayer...etc).

Lighting

Pour sublimer l'environnement nous avons fait usage de lighting. Le lighting apporte lui aussi certaines contraintes, notamment le nombre de lumières pouvant être affichées par objet, ce qui nous a forcé à diviser notre environnement en un grand nombre d'objets afin de pouvoir gérer les lumières à notre guise et réaliser un éclairage des plus adapté. Nous avons dû faire face à un autre problème, les lights, si on souhaite les baker utilisent les maps UV des objets, or, nous n'avons pas UV l'environnement suite au choix du triplanar.

Nous avons finalement réussi à rendre correctement les lights en utilisant la feature « generate lightmap UVs » d'Unity, permettant donc de créer des maps d'UV sur des objets qui n'en ont pas pour s'en servir pour le lighting.

L'environnement de notre jeu comprend une partie en extérieur et nous souhaitions, pour des raisons de cohérence séparer le lighting intérieur de celui à l'extérieur. Nous avons donc créé un grand nombre d'objets dits « Lightblockers » afin d'empêcher la lumière de dehors de rentrer à l'intérieur. Par extension et soucis du détail nous avons voulu pousser le réalisme un peu plus loin en ajoutant des verrières dans les light blockers afin de faire passer de la lumière issue de l'extérieur à l'intérieur, comme le ferait une vraie verrière située au plafond.

De plus, un soleil dynamique a été intégré ce qui nous permet d'avoir des inclinaisons d'ombres très proches du réel, qui complètent et subliment l'utilisation de nos diverses verrières. Cet outil recréé la position en temps réel du soleil en se basant sur des coordonnées géographiques du monde réel. Un outil fort amusant ajouté par notre programmeur !

Toon Shader

Lambert

Tout d'abord, nous obtenons la lumière via la formule de la source lumineuse lambertienne.

Diffuse = N dot LightDir * LightIntensity

Step

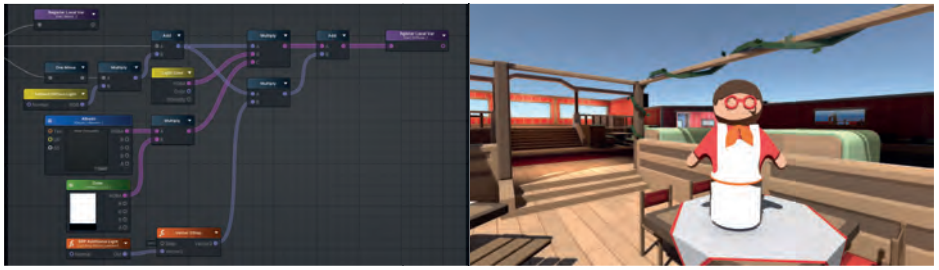
Ensuite, pour obtenir le côté flat, nous effectuons un step sur le lambert.

Rim Lighting

N'importe quel shader toon mérite du rim lighting. Pour ce faire, nous utilisons un effet de Fresnel.

Specular

Ensuite, nous calculons la specular du lighting via la direction de la view et la direction du lighting.



Color

Avant le rendu final, nous calculons la couleur du personnage. Tout d'abord via l'ajout de la diffuse indirecte. Après nous ajoutons une texture pouvant être teinte par une couleur.

Après, nous ajoutons les lumières additionnelles aux couleurs pour que nos personnages soient éclairés par d'autres lumières que seul la directional.

Lighting

Soleil Dynamique

Pour rendre l'éclairage de notre environnement plus dynamique, nous avons développé une simple simulation de l'orientation du soleil en fonction du moment où joue le joueur.

Pour ce faire nous avons donc fait des recherches sur le soleil et nous avons trouvé l'article "A solar azimuth formula that renders circumstantial treatment unnecessary without compromising mathematical rigor: Mathematical setup, application and extension of a formula based on the subsolar point and atan2 function" de Taiping Zhang, Paul W. Stackhouse, Bradley Macpherson, et J. Colleen Mikovitz.

Il se base sur le point subsolaire et la fonction atan2. Il présente des calculs et formules nécessaires pour obtenir l'azimuth du Soleil, de manière mathématique et en Fortran.

Pour ce faire, nous devons calculer plusieurs formules.

Les étapes du calcul :

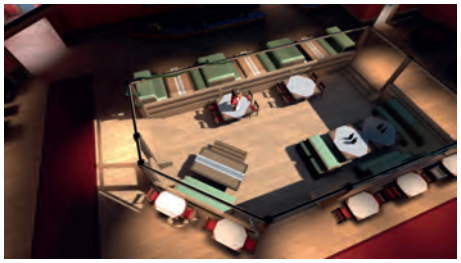
1. Nombre de jours en Terrestrial Time (J2000.0 UT)
2. La longitude moyenne du Soleil
3. L'anomalie moyenne du Soleil
4. La longitude écliptique du Soleil
5. L'obliquité de la Terre
6. L'ascension droite du Soleil
7. La déclinaison du Soleil
8. L'équation du temps
9. La latitude et longitude du point subsolaire.
10. L'angle d'azimut du soleil.

Avec ces valeurs, il est donc possible d'obtenir la direction du Soleil à un moment et à un point donné.

Coordonnées

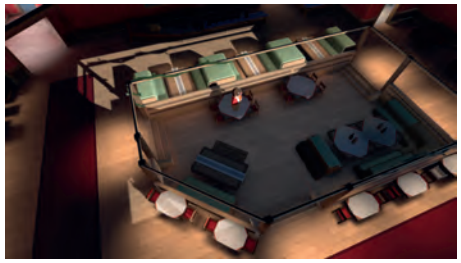
Les coordonnées que nous utilisons sont :

48,855201°N, 2.272845°E



Différences

La capture d'écran ci-dessus simule la date : 01/06/24 10h40m17s.



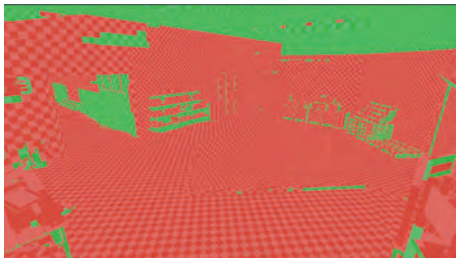
Différences

La capture d'écran ci-dessus simule la date : 01/11/24 10h40m17s.

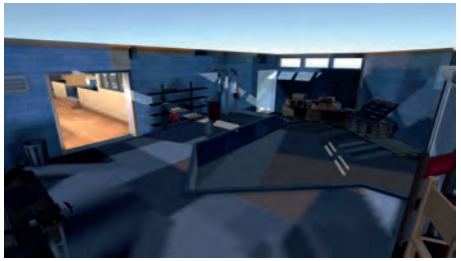
Validité des Texels

Un problème que nous avons rencontré lors de notre lighting pass est que l'ajout de lightblockers avait totalement cassé les lightmaps. C'est à dire que les lightmaps bake se superposaient en intérieur.

Après l'utilisation des différents modes de rendus de la vue scene d'Unity, nous avons remarqué que la plupart des texels de l'intérieur étaient invalides.

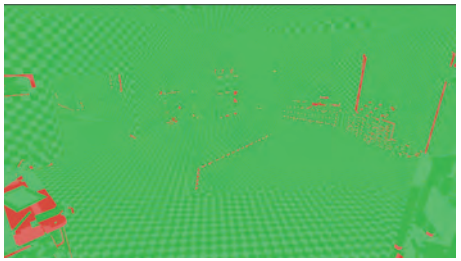


Chaque carrée représente un texel. Un texel vert est valide et n'est pas overlap par un autre, tandis qu'un rouge est invalide et est overlap lors du bake.



Après quelques recherches, nous avons découvert qu'un texel est marqué comme invalide par Unity s'il peut voir la backface d'un élément. Ce qui est logique : si il voit une backface c'est qu'il est probablement obstrué par de la géométrie.

La solution était donc d'appliquer un material double sided sur les blockers n'ayant pas de face vers l'intérieur.



Une fois que la solution a été appliquée, il n'y avait plus aucun problème de lighting. Nous avons donc pu reprendre nos travaux de lighting.

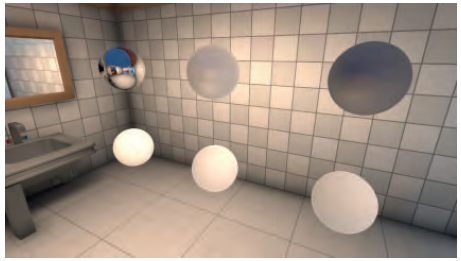
Light Probes

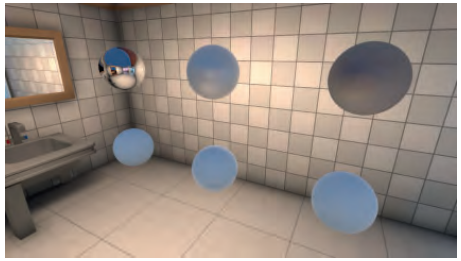
Le light probe dans Unity est un component affectant la lumière des objets dynamiques. L'objectif de ces dernières est d'appliquer la bake GI faite à l'edit time des objets static sur des objets dynamique au runtime.

Ainsi, ces objets permettent de bénéficier de lighting de haute qualité sur des objets dynamiques. Ce qui apporte l'avantage d'avoir un lighting bake mais avec des objets dynamiques.

Les deux screenshots suivant représente la même pièce, mais la seconde ne possède pas de light probe.

On remarque que la seule lumière reçue par les sphères est la lumière ambiante de la skybox de la scène, et non la lumière jaunâtre de la pièce.

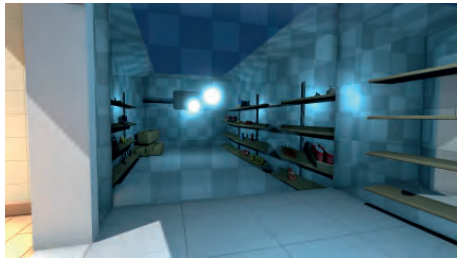


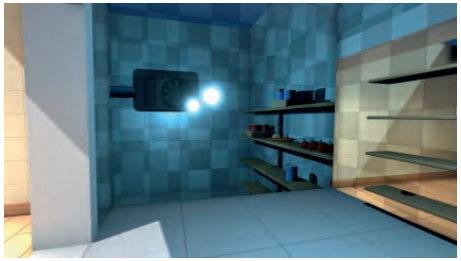


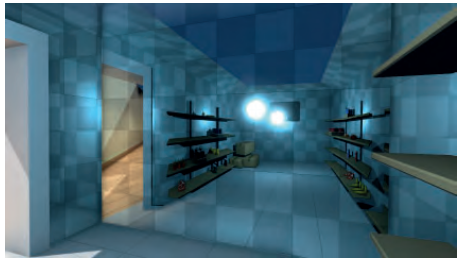
Reflection Probes

Comme leur noms l'indique, les reflections probes sont des objets gérant les reflexions. Elles sont bake à l'edit time.

La plupart de nos reflections probes sont parralax corrected. C'est à dire qu'elles vont effectuer un effet de parallaxe pour aligner les réflexions avec le point de vue de la caméra.







Cubemap

La réflexion de cette cubemap n'est pas corrigé par la parallaxe. Ce qui donne un effet bizarre sur les surfaces réflectives, car elles restent fixées sur la surface.

Box Cubemap

Les deux images ci-dessus montrent l'effet d'une cubemap corrigé par l'effet de parallaxe. La réflexion suit la caméra, ce qui rend la réflexion plus réelle.

VFX : Smoke

Introduction

La Fumée

Le VFX de fumée est une composante importante dans notre projet permettant d'apporter un feedback important pour la feature centrale de catastrophe. Cette documentation détaille l'implémentation de cet effet dans notre projet.

Objectifs

Esthétique Réaliste : Recréer une fumée réaliste et détaillée pour renforcer l'immersion des joueurs.

Animation Dynamique : Intégrer des animations de fumée dynamiques pour simuler le mouvement fluide et naturel de la fumée.

Feedback Efficace : Implémenter la fumée de manière à ce qu'elle puisse être modifiée en temps réel pour correspondre aux différentes situations de jeu.

Méthodologie

Recherches/Moodboard

Rassembler des images, des vidéos et des œuvres d'art représentant différentes formes de fumée dans diverses conditions.

Observer des phénomènes réels tels que des incendies, des explosions, et de la fumée industrielle pour comprendre les variations de densité, de couleur et de comportement de la fumée.

Analyse Technique

Nous avons étudié les techniques utilisées dans les films et les jeux vidéo pour simuler la fumée.

Prototypage Rapide

Nous avons opté pour Blender et Shadergraph pour le prototypage de ce VFX.

Tests Utilisateurs

Partage des prototypes avec d'autres membres de l'équipe ou des utilisateurs.

Itération

Basée sur les retours des tests utilisateurs et les exigences du design du jeu, affiner les paramètres de la simulation, ajuster les textures, et améliorer les shaders.

Répéter le processus de test et de retours jusqu'à obtenir une version satisfaisante de l'effet de fumée.

Implémentation dans le projet

Utiliser des scripts C# pour contrôler les effets de fumée en temps réel et s'assurer qu'ils répondent de manière dynamique aux événements de jeu.

Recherches

Couleur et Transparence

La couleur de la fumée peut varier en fonction des conditions (par exemple, fumée noire pour les incendies, blanche pour la vapeur).

Utiliser des dégradés de couleurs et des textures semi-transparentes pour simuler les variations de densité et d'opacité.

Comportement Dynamique

La fumée doit réagir aux forces externes comme le vent et les collisions avec les objets environnants.

Intégrer des animations de turbulence et de dissipation pour simuler le comportement naturel de la fumée au fil du temps.

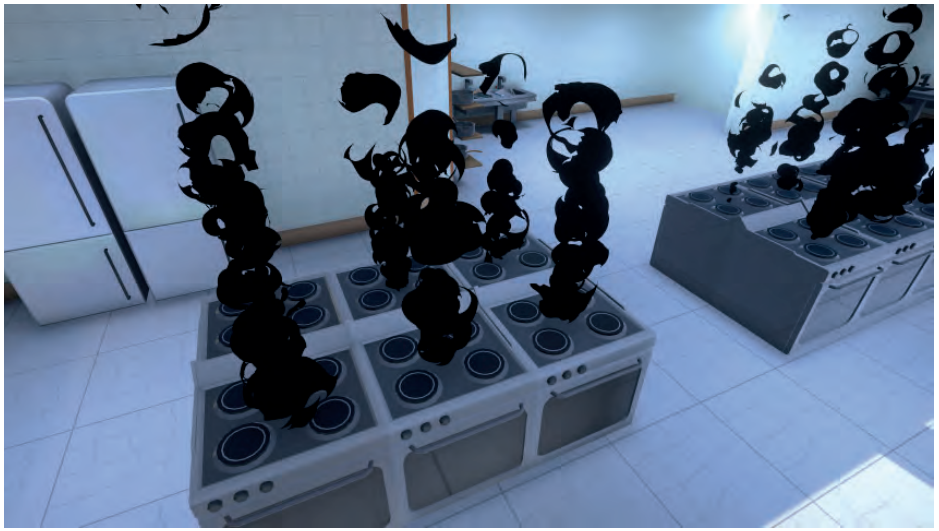
Dégradation

Simuler la dégradation naturelle de la fumée, en la faisant s'estomper et se disperser progressivement.

Ajouter des détails comme des particules de cendres ou des débris pour augmenter le réalisme.

Qualité

Nous utiliserons potentiellement le node screen space afin de contrôler plus efficacement le niveau de qualité par rapport à l'écran.



Recherches Effet Smoke


1


2


3

1. Suspension (your disappearing one) **Peter Roux**
2. *The Sound of Music* **Javiera Estrada**
3. *Sculpting with smoke* **Iryna Nalyvaiko**
4. *The smoky boy* **Yannick Duriez**
5. *California* **Claire Cansick**

Commentaires

Caractéristiques Visuelles

J'ai pu en extraire des informations sur la densité et le volume me permettant d'itérer dans mes créations.

Il y a aussi un questionnement sur l'opacité et la fumée et ses variations.

On constate que les bords des nuages irréguliers et arrondis.

Il faut des formes irrégulières et variées pour éviter une apparence trop uniforme.

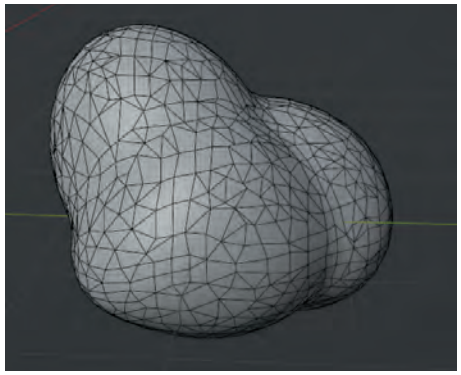

4

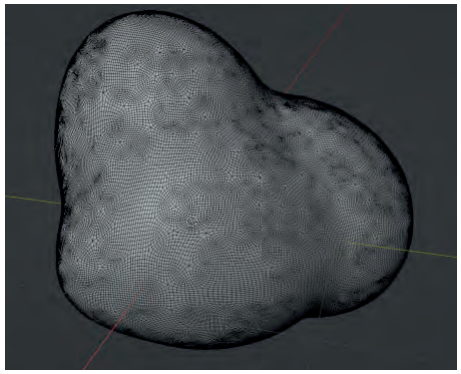

5

Analyse & Prototypage

Volume de la fumée

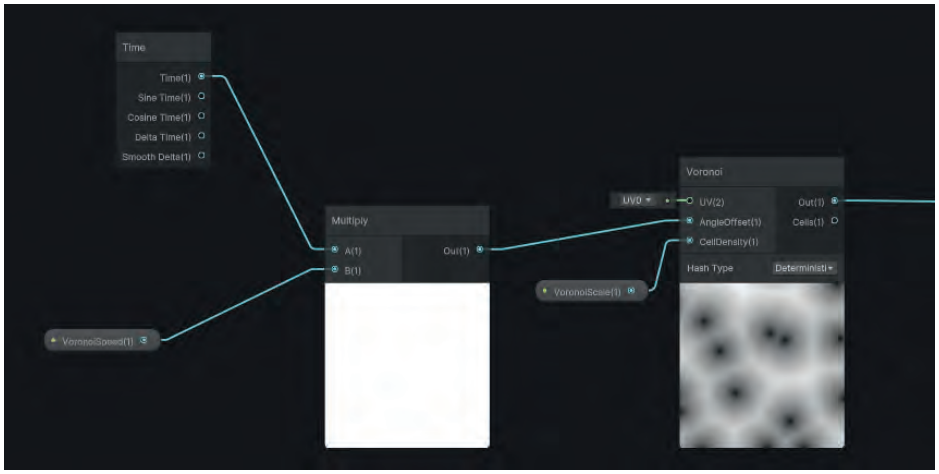
- Création d'un mesh représentant de le volume de la fumée sur blender grâce à un process en subdivision.
- Ensuite optimisation du mesh afin de le rendre game ready.
- Importation du mesh dans unity





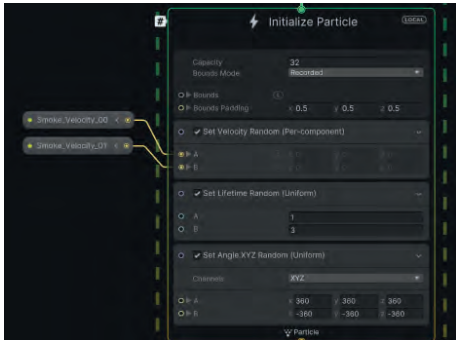
Matériau de la fumée

Afin de produire un effet de désintégration de fumée, j'ai opté pour un texture procédurale de voronoï permettant de désintégrer la fumée en temps réel selon un pattern de cellules.



Particule System

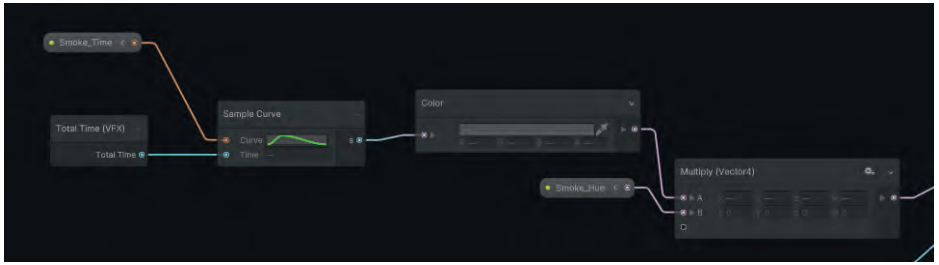
Setup du particule system pour paramétrer la vélocité des particules. On fait en sorte de randomiser la durée de vie et l'angle des particules.



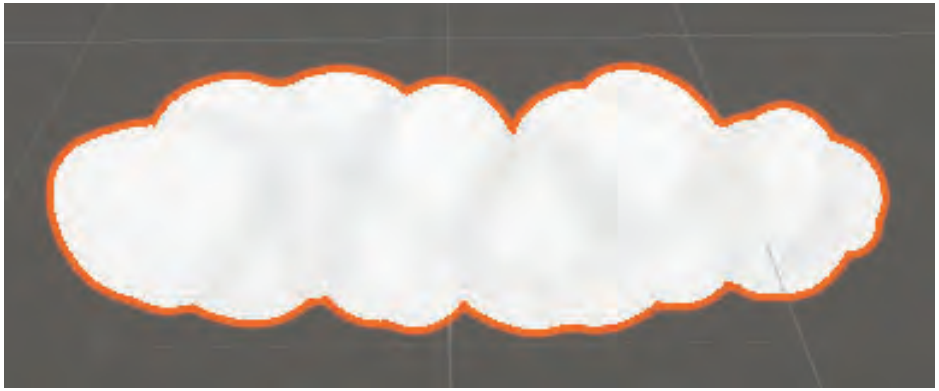
Implémentation de la feature

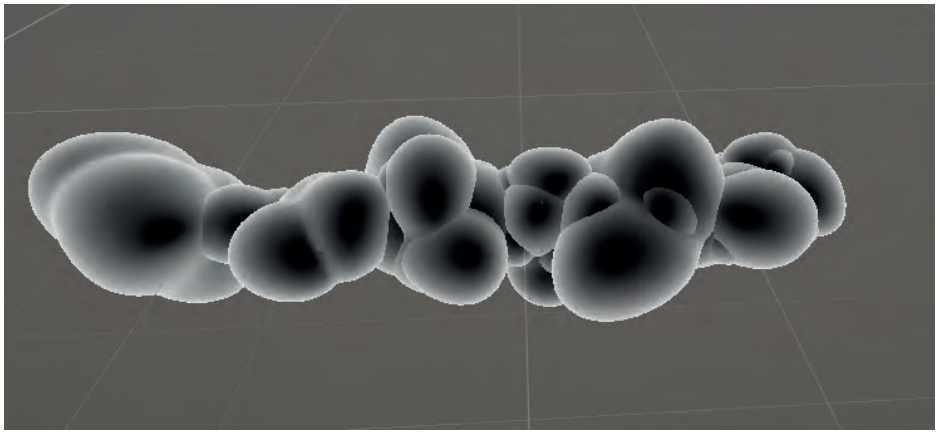
Ajout d'une courbe permettant de contrôler à quel point est-ce que la fumée noircit au cours du temps.

On peut aussi contrôler la couleur et la taille au cours du temps.



Rendu à différentes étapes du prototypage

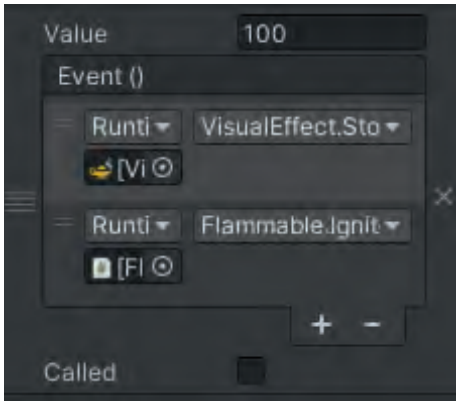


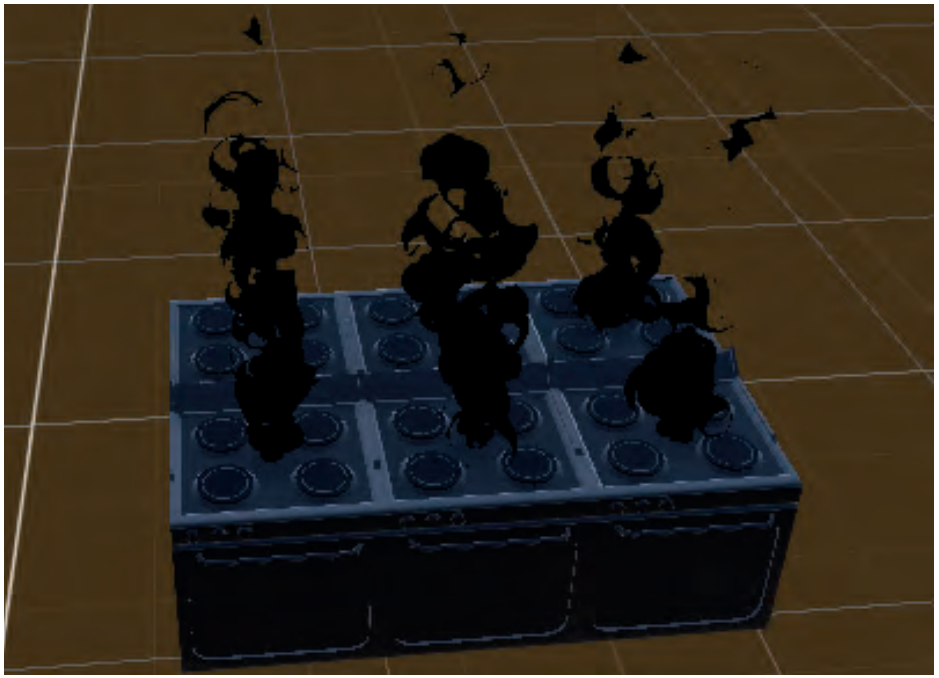


Implémentation

Afin d'implémenter ce feedback nous utilisons des Prefabs appelé par des events

La fumée est appelée quand on fait cuire un objet, après un certain temps comme on peut le voir à droite le script est appelé si l'objet en train de cuire n'est pas retiré





Introduction

Des Flammes ?

Un VFX de flammes est un composant essentiel de notre projet, offrant un feedback visuel impressionnant pour illustrer la centralité de la catastrophe. Cette documentation détaille l'implémentation de cet effet dans notre projet.

Objectifs

Intégrer des animations de flammes dynamiques pour simuler le mouvement fluide et naturel du feu

Implémenter les flammes de manière à ce qu'elles puissent être modifiées en temps réel pour correspondre aux différentes situations de jeu.

Méthodologie

Recherches/Moodboard

Rassembler des images, des vidéos et des œuvres d'art représentant différentes formes de flammes dans diverses conditions.

Afin de comprendre les variations de couleur, d'intensité et de comportement des flammes.

Analyse Technique

Nous avons étudié les techniques utilisées dans les films et les jeux vidéo pour simuler la fumée.

Prototypage Rapide

Nous avons opté pour Illustrator et Shadergraph pour le prototypage de ce VFX.

Tests Utilisateurs

Partage des prototypes avec d'autres membres de l'équipe ou des utilisateurs.

Itération

Basée sur les retours des tests utilisateurs et les exigences du design du jeu, affiner les paramètres de la simulation, ajuster les textures, et améliorer les shaders.

Répéter le processus de test et de retours jusqu'à obtenir une version satisfaisante de l'effet de fumée.

Implémentation dans le projet

Utiliser des scripts C# pour contrôler les effets de fumée en temps réel et s'assurer qu'ils répondent de manière dynamique aux événements de jeu.

Recherches

Couleur et Transparence

La couleur de la fumée peut varier en fonction des conditions (par exemple, fumée noire pour les incendies, blanche pour la vapeur).

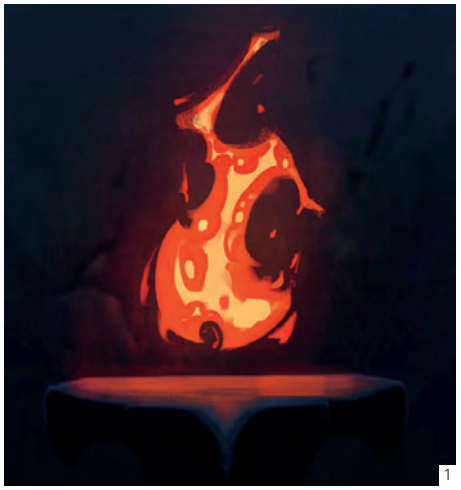
Utiliser des dégradés de couleurs et des textures semi-transparentes pour simuler les variations de densité et d'opacité.

Comportement Dynamique

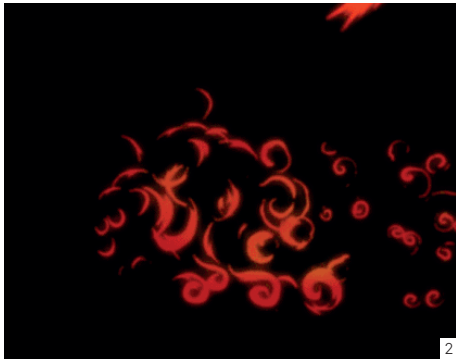
Intégrer un mouvement ascendant pour simuler le comportement naturel des flammes au fil du temps.

Dégradation

Simuler la dégradation naturelle des flammes en les faisant s'estomper et se dissiper progressivement.


1


3


2

1. *Studies* **Joshua Adegboye**
2. *f+s+flame* **enfanterrible**
3. *Lightning vfx Unity* **Duc K**

Commentaires

Caractéristiques Visuelles

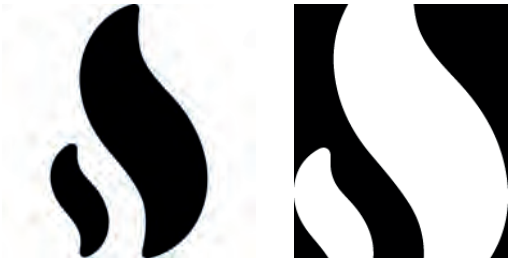
Avec cet échantillon d'images, nous avons compris que notre VFX de flammes devrait avoir plusieurs caractéristiques très précises telles que l'intensité lumineuse, la présence d'étincelles ainsi que le fade out de la luminosité des flammes du foyer s'estompant peu à peu avec la hauteur.

Il était important pour nous de trouver des références aux effets "cartoon" afin de coller avec l'esthétique de notre direction artistique.

VFX : Fire

Analyse Technique & Prototypage

Texture de flamme



J'ai d'abord commencé par créer un sprite de flamme sur Illustrator. Puis, j'ai créé une forme simple évoquant une flamme en vectoriel sur un fond noir.

Afin de pouvoir facilement utiliser l'alpha clip de Unity.

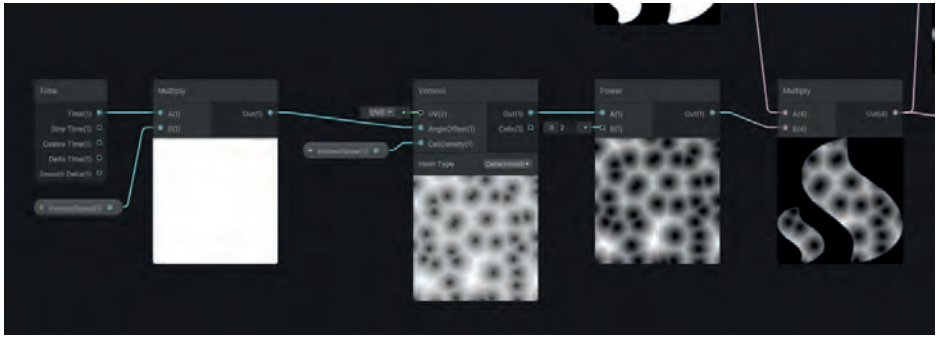
Shader de flamme

Ensuite, j'ai commencé l'élaboration un shader permettant de faire varier la texture au cours du temps.

En effet ce shader permet grâce à une texture procédurale en voronoï et donner cet effet de dissipation et de varier que l'on peut retrouver dans une flamme

J'ai pris soin de faire en sorte grâce au fait que ma flamme soit blanche et mon fond soit noir que seul les flammes soit déformées par la textures.

On peut ensuite y ajouter utilement une couleur ainsi qu'une intensité.



Particule System

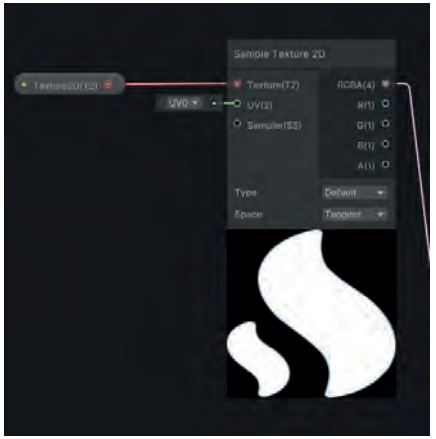
Ensuite, j'ai setup un VFX qui spawn des flammes constamment.

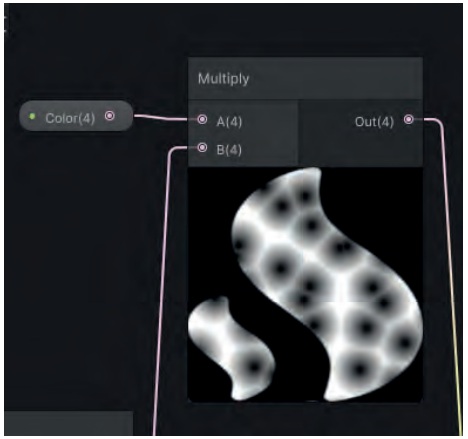
Selon un angle et une durée de vie aléatoire.

Ses particules ont le shader crée précédemment nous permettant de configurer les valeurs dans les VFX afin de faciliter l'itération de faire varier certains paramètre tel que l'alpha clip.

Qui lui permet que le feu s'estompe complètement au bout d'un certain temps. On peut gérer la

vélocité du feu, à quel point il est dense, sa couleur, et sa taille au cours de sa vie.

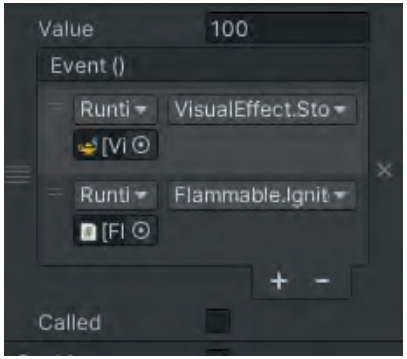


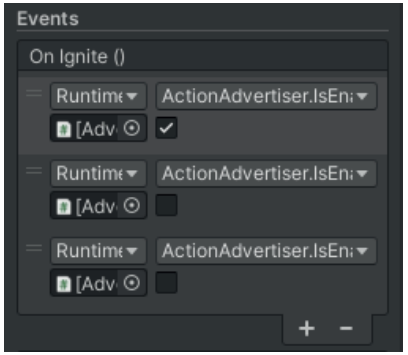


Implémentation de la feature principale

L'ajout d'une courbe permettant de contrôler à quel point est-ce que la fumée noircit au cours du temps.

On peut aussi contrôler la couleur et la taille au cours du temps.





IA : Conception

Objectifs et modèles adaptés

La première étape de notre processus fut d'établir nos objectifs quant à l'utilisation de l'IA. Nous avons décidé d'utiliser cet outil afin de générer des concepts arts. Cependant, nous souhaitions aller plus loin en expérimentant l'habillage de blocking de level design grâce à des méthodes d'image to image.

Suite à la définition de nos objectifs, nous avons fait une veille sur les différentes technologies d'IA qui nous semble pertinente.

Nos critères de recherche principaux sont de ne pas être limité dans son utilisation, la qualité des outputs ainsi que la modularité des paramètres liés à la génération. Ceci dans l'optique d'utilisé une technologie d'image to image bien moins abordable et facile d'utilisation que le classique texte to image que le grand publique à l'habitude d'utiliser.

Ainsi, nous nous sommes arrêtés sur deux IA principales qui semblent complètement correspondre à nos besoins.

Nos recherches se sont arrêtées sur deux IA :

-Bing Image Creator, une version de DALL-E disponible en ligne gratuitement et sans réelle limite. Nos précédentes utilisations nous ont permis de déceler son potentiel sans avoir besoin d'une trop grande maîtrise de l'outil. Ainsi, avec une bonne utilisation, nous pensons pouvoir obtenir des résultats facilement pertinents.

-Stable Diffusion, est une IA que nous maîtrisons tout particulièrement. Elle est open source et permet une grande liberté quant aux paramètres de génération. De plus avec Stable, nous pouvons installer une infinité de modèles différents et de différents types.

Définition modèles

Le modèle contient les données qui serviront à générer l'output. Les modèles sont la base des réseaux neuraux génératifs. Stable diffusion, Dreamshaper, Midjourney sont des modèles. Dans le cas d'IA générative d'images, ils peuvent être génériques ou plus efficaces pour un thème ou un style en particulier, car ils auront été entraînés spécifiquement pour cela. Il existe différents types de modèles dont l'implication dans le processus génératif varie.

Parmi eux : les checkpoints sont des modèles « généraux » qui permettent de générer seul des images. Les modèles officiels de Stable sont des checkpoints par exemple, mais n'importe qui peut en entraîner.

On retrouve aussi des modèles « Lora » qui sont entraînés sur un aspect plus précis de la génération. Cela peut être un style, une position, un personnage en particulier... Ils s'utilisent de pair avec un checkpoint.

Technique ControlNet

ControlNet est un modèle fonctionnant de paire avec une autre modèle checkpoint. Durant notre projet, nous l'avons utilisé avec Stable Diffusion sur l'interface AUTOMATIC 1111.

ControlNet permet de forcer la génération pour obtenir une composition d'image ou une position précise. Le tout, en utilisant à la fois un prompt textuel basique et en complémentarité d'un prompt image.

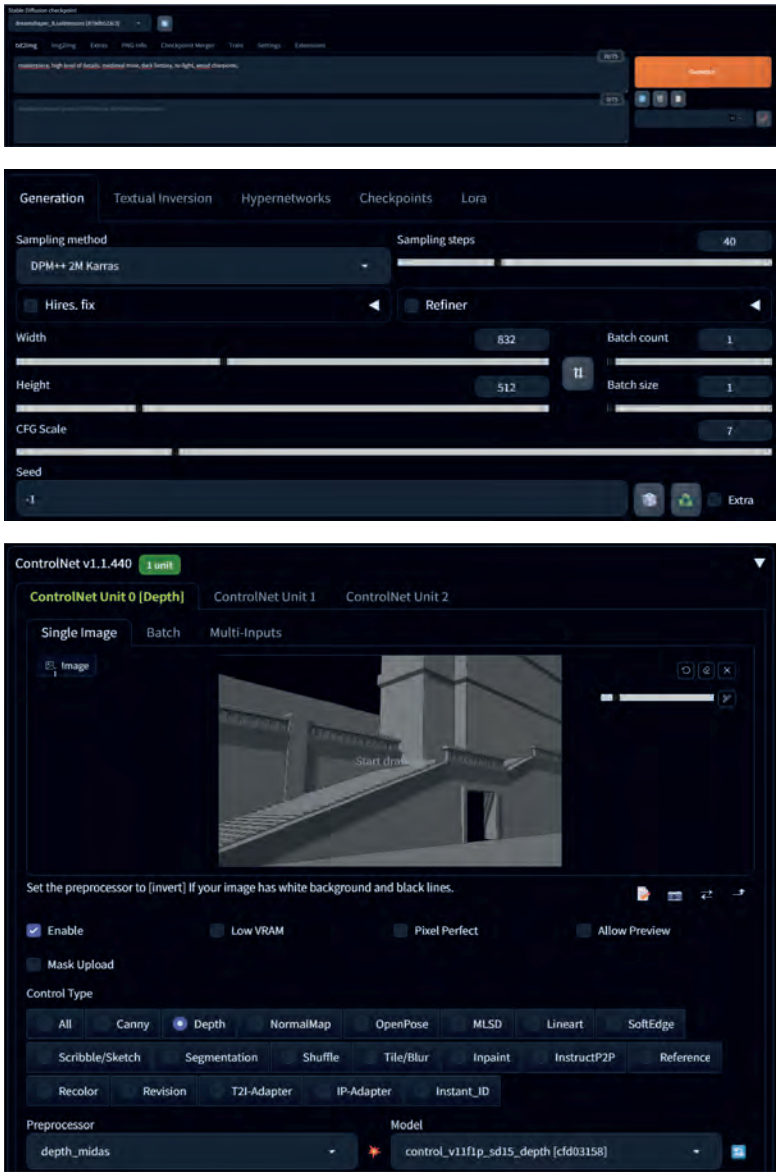
ControlNet est donc un outil d'image-to-image permettant de générer avec une précision parfaite. Il semble être une réponse parfaite pour répondre à un problème endémique des IA, à savoir la difficulté de générer des images précises et cohérentes.

Les IA standards de génération d'images offrent peu de contrôle sur la composition finales des images. L'output est inévitablement aléatoire et pour avoir des résultats intéressants, il est inévitable d'avoir une approche itérative ControlNet permet d'avoir une approche bien moins chronophage dans nos démarches. C'est d'ailleurs notamment grâce à ControlNet que les gens réalisent des images avec du texte ou des QR Code cachés dedans.

ControlNet est donc une IA utilisée en extension avec des modèles de Stable Diffusion. Mais il a aussi ses propres modèles qui permettent d'extraire différentes informations des images que l'on donne en input. Cela peut être des positions, une profondeur d'image, les contours ou les traits, et même des normals maps. Chacune de ses possibilités vient avec son lot de modèle propre à ControlNet permettant d'obtenir certes le même type d'information, mais de manière différente.

De plus dans le projet, nos shaders permettent d'avoir un mode débug qui affiche l'orientation de chaque face. C'est pour cela que nous avons fini par l'utiliser pour permettre à ControlNet de mieux capter les angles de nos blocking. Nous avons découvert cette technique par hasard en recherchant un moyen d'éviter que ControlNet ne détecte plus les textures de mesure que les angles du blocking directement.

IA : Conception

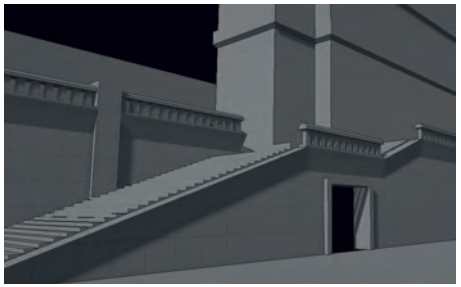


UI AUTOMATIC 1111 pour Stable Diffusion avec l'extension ControlNet d'installé

Une fois l'extension installée, on peut de nouveau, comme sur Stable Diffusion, jouer avec tout un tas de paramètres. Ici, les principaux sont l'image d'input, le type d'output que l'on souhaite extraire avec ControlNet, les pré-processeurs et les modèles. Le type d'output est donc la liste de points que l'on peut sélectionner comme Canny, Depth, NormalMap etc...

Ensuite, pour simplifier, un préprocesseur lit les données qui seront utilisées par le modèle ControlNet sélectionné. Ainsi, chaque modèle a besoin d'un préprocesseur précis. Un modèle utilisé pour obtenir les informations de depth n'utilisera pas le même préprocesseur qu'un modèle utilisé pour obtenir la position d'un personnage sur une image.

Les modèles sont des IA intermédiaires qui génèrent une image à mi-chemin qui sera lu par ControlNet pour diriger la génération de l'image finale. Ainsi, il y a différents modèles de NormalMap par exemple qui donne donc des normals maps plus ou moins diffusent. Certains modèles sont plus diffus et captent la totalité de l'image, alors que d'autre sur les éléments du premier plan qui ressortent clairement.



IA : ControlNet

ControlNet propose donc plusieurs types de détection, la première étant la détection Canny, un algorithme conçu en 1986 par John Canny. Bien que la technique soit vieille, son utilisation reste pourtant bien actuelle dans le cadre de l'utilisation des IA. Ensuite, nous pouvons extraire des Depth maps et des normals maps qui permettent de repérer les différents plans dans la composition de notre image.

En plus de tous ces types de détection, ControlNet permet aussi de détecter les positions de corps humain, de doigts et de visage, mais nous ne l'avons pas utilisé dans notre projet. De plus ControlNet permet d'inpaint pour forcer le noise initiale d'une image à être ignoré. De cette manière, on conserve une forte variation sur l'image excepté à certains endroits bien choisis.

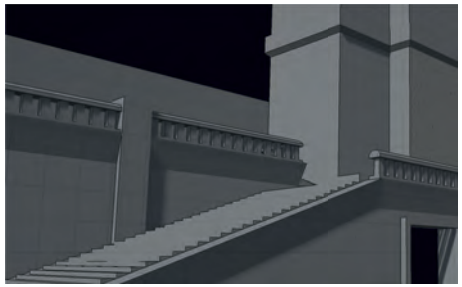
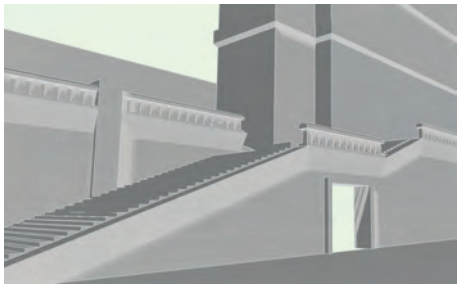
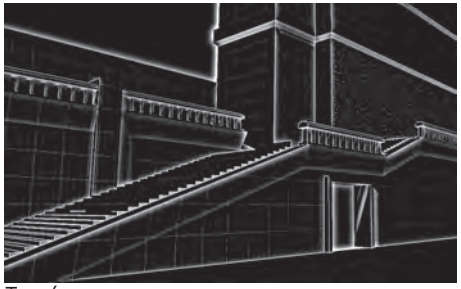


Image d'input



Contours

Normal map



Tracés

Depth map

Egyptian architecture

Modern House

Asian palace

Medieval mine

ControlNet permet d'itérer à une vitesse folle en conservant la même composition d'image. Il nous paraît plutôt évident qu'une utilisation maîtrisée permettrait de donner des résultats rapides et pertinents dans le cas d'une production professionnelle.

IA : ControlNet

Utilisation multi modèle

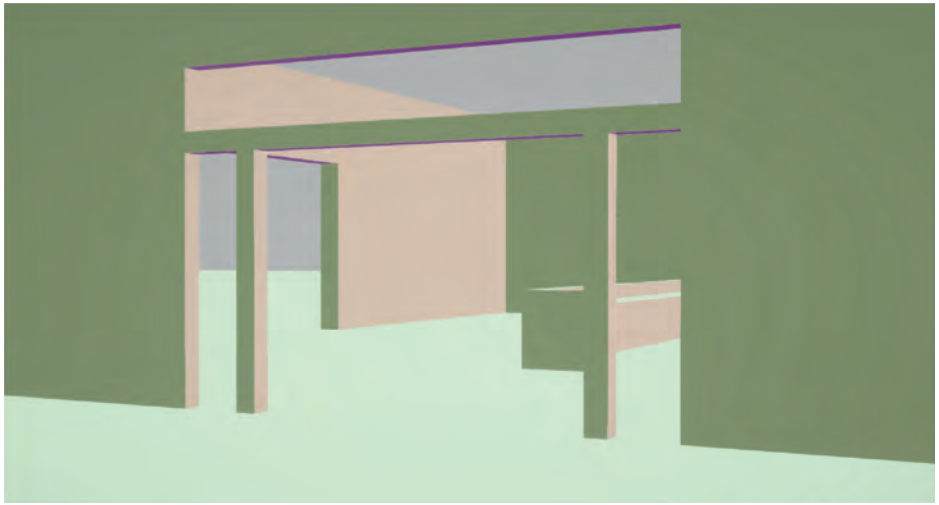
Avec toutes ses techniques maîtriser, il est désormais possible d'utiliser plusieurs modèles. Ainsi, pour tester de stylisé nos outputs et obtenir des résultats moins réalistes.

C'est dans cette optique que nous avons utilisé plusieurs modèles afin de générer des images dans un style plus facilement. Le tout, en utilisant ControlNet nous avons pu essayer de styliser et d'habiller certains de nos blocking. Nous avons utilisé le checkpoint de Stable Diffusion XL combiné avec un Lora trouvé sur le site Civit.ai (Lien https://civitai.com/models/285077/black-and-white-geometric).

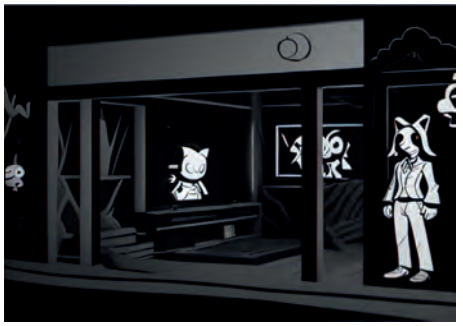
De cette manière, nous avons fusionné des outils qui permettent d'avoir des outputs précis comme ControlNet et le checkpoint de Stable Diffusion XL qui est plutôt performant avec un Lora qui instaure un stylé précis, mais aussi plus d'aléatoire.

Il en résulte quelque chose de plus intéressant en termes de rendu. Dans nos recherches en direction artistique cela nous à permis de rapidement itérer et tester des rendus quasiment finis possibles.

Cependant, au fur et à mesure que nos intentions se précisaient, nous avons fini par rechercher un Lora capable d'aiguiller la génération sur une image avec de la outline. Mais malgré nos recherches et nos tests, nous n'avons trouvé aucun Lora performant dans cette tâche.

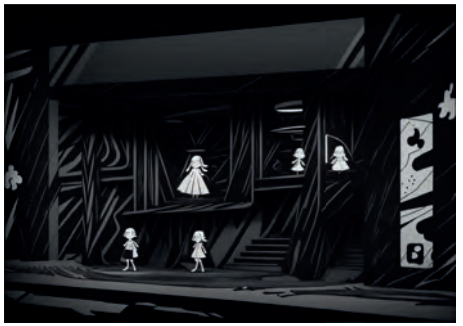


Ci-dessus l'image de base avec un rendu de debug permis par les shaders du projet pour permettre à ControlNet de mieux capté les angles



1

2



3

4

1 : masterpiece, toonshader, black outline, shader outline, video games concept art, hidden folks, genesis noir, <lora:BnW-abstract-128dim-v1:1>, Steps: 20, Sampler: DPM++ 2M Karras, CFG scale: 7, Seed: 2888253024, Size: 728×512, Model hash: 879db523c3, Model: dreamshaper_8, ControlNet 0: «Module: canny, Model: control_v11p_sd15_canny

2 : masterpiece, toonshader, black outline, white texture, shader outline, video games concept art, hidden folks, genesis noir, <lora:BnW-abstract-128dim-v1:0.8>, mall interior, Steps: 20, Sampler: DPM++ 2M Karras, CFG scale: 7, Seed: 2888253024, Size: 728×512, Model hash: 879db523c3, Model: dreamshaper_8, ControlNet 0: «Module: canny, Model: control_v11p_sd15_canny

3 : masterpiece, toonshader, black outline, shader outline, video games concept art, hidden folks, genesis noir, <lora:BnW-abstract-128dim-v1:1>, Steps: 20, Sampler: DPM++ 2M Karras, CFG scale: 7, Seed: 2888253024, Size: 728×512, Model hash: 879db523c3, Model: dreamshaper_8, Denoising strength: 0.7, ControlNet 0: «Module: canny, Model: control_v11p_sd15_canny

4 : masterpiece, toonshader, black outline, white texture, shader outline, video games concept art, hidden folks, genesis noir, mall interior, Negative prompt: character, Steps: 20, Sampler: DPM++ 2M Karras, CFG scale: 7, Seed: 4172198401, Size: 728×512, Model hash: 879db523c3, Model: dreamshaper_8, ControlNet 0: «Module: canny, Model: control_v11p_sd15_canny

IA : Character Design

Les biais des IA

Dans le but de faire du concept art par IA, nous avons aussi voulu exploiter les biais qu'on les IA. C'est pour cela que nous avons tenté de générer des « métiers» pour des personnages puis constaté les habits qui étaient le plus présent dans les outputs.

Étant donné que nous avions déjà une idée précise de l'apparence qu'auraient nos personnages, il ne nous semblait pas pertinent de générer des concept art pour ceux-ci. Cependant, l'habillage de ceux-ci est un pan important de notre design puisque le joueur est censé rapidement comprendre la tâche de chaque personnage en un coup d'œil.

Grâce à ce travail, nous avons pu établir des habits « type » pour chacun de nos personnages. De plus, nos rendus essayent de se rapprocher le plus du style de nos personnages afin d'avoir des éléments de repère au point de jonction notamment (coup, bras, etc...)

1

2

3

4

5

1 : 3D video game item concept art, video game character, neutral background, (human fall flat video game style), no details, goofy style, no shadows, technician operator, focus on one item

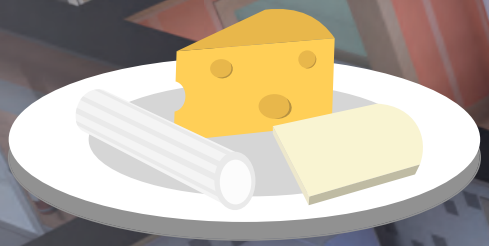
2 : 3D video game item concept art, video game character, neutral background, (human fall flat video game style), no details, goofy style, no shadows, bartender, focus on one item

3 : 3D video game item concept art, video game character, neutral background, (human fall flat video game style), no details, goofy style, no shadows, cleaning man, focus on one item

4 : 3D video game item concept art, video game character, neutral background, (human fall flat video game style), no details, goofy style, no shadows, server, focus on one item

5 : 3D video game item concept art, video game character, neutral background, (human fall flat video game style), no details, goofy style, no shadows, a cook, focus on one item

Sound Design



Fromages

Intentions & Références

Fmod

Documentation

Mécaniques

Prise de son

Musique

Introduction

Itentions sonores

This Was Fine est un jeu de type sandbox, où le joueur, faisant face à un système résorbant, devra trouver des moyens de créer du chaos et d'atteindre des objectifs de désordre. On souhaitait faire ressentir au joueur son impact sur l'environnement et sa capacité à modifier le système. Il était par conséquent important que nos sons soient en accord avec cette intention de départ.

Notre niveau prend place dans un restaurant, et nous y avons aussi construit une ambiance sonore afin de rendre l'expérience du joueur plus immersive et agréable. Nos sons sont principalement découpés en quatre catégories :

- les sons d'environnement
- les sons d'interface produits par le joueur
- les sons produits par les NPC
- les sons produits par les items du niveau

Nous voulions apporter une atmosphère goofy et rassurante qui est souvent caractéristique de ce type de jeu. Par conséquent, nous avons travaillé sur des feedbacks doux et humoristiques.

Le son a été travaillé en adéquation avec les feedbacks visuels pour avoir la meilleure cohérence possible, tout en signifiant au joueur les informations importantes. En effet, lors d'une partie, il se passe beaucoup de choses à l'écran, et le joueur doit avoir un retour sur chaque action, mais sans pour autant le submerger de feedbacks sonores. Notre intention était de créer une ambiance sonore dynamique et réactive qui évolue en fonction des actions du joueur.

Chaque interaction, qu'il s'agisse de déclencher un incident ou de provoquer une réaction des NPC, est accompagnée de sons distincts et percutants qui accentuent le caractère ludique et désordonné du jeu. Les effets sonores sont conçus pour être à la fois humoristiques et énergiques, capturant l'essence de la pagaille orchestrée par le joueur. Par exemple, déclencher un incendie ou provoquer une fuite d'eau ne se contentera pas d'être visuellement spectaculaire, mais sera accompagné de sons vifs et exagérés, accentuant le côté absurde et hilarant de la situation.

En outre, nous avons intégré une bande sonore rythmée et effervescente qui intensifie l'urgence et le chaos, tout en maintenant une atmosphère légère et divertissante. Ainsi, notre sound design vise non seulement à soutenir le gameplay, mais aussi à enrichir l'expérience globale en rendant chaque action plus tangible et gratifiante. Ainsi, chaque action, aussi chaotique soit-elle, devient une partie intégrante de la symphonie désordonnée qui définit notre jeu.

Références sonores

Pour le développement sonore de notre jeu, nous nous sommes inspiré de références notables telles que Human Fall Flat, Overcooked 2, Wobbly Life et Untitled Goose Game. Ces choix ne sont pas faits au hasard ; chacun de ces jeux incarne des aspects spécifiques du sound design que nous souhaitons intégrer et adapter à notre propre univers.

Human Fall Flat est intéressant pour son utilisation intelligente des sons pour accentuer l'humour physique et les interactions imprévisibles entre les personnages et l'environnement. Les effets sonores y sont simples mais très efficaces, soulignant chaque mouvement maladroit avec des bruits distincts qui ajoutent à l'expérience comique et immersive.

Overcooked 2 excelle dans la création d'une atmosphère frénétique et énergique. Les sons de cuisine, les cris des personnages et la musique rythmée travaillent ensemble pour maintenir un sentiment constant d'urgence et de chaos contrôlé. Cette synergie sonore est précisément ce que nous voulons reproduire dans notre jeu pour que chaque situation semble à la fois pressante et hilarante.

Les effets sonores de Wobbly Life ajoutent de la profondeur et du réalisme aux interactions avec l'environnement tout en conservant une tonalité légère et amusante, ce qui est essentiel pour notre jeu où le joueur doit semer la pagaille.

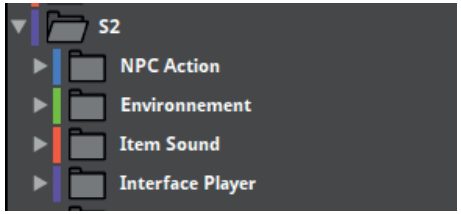
Enfin, Untitled Goose Game est très fort en matière de sound design minimaliste et efficace. La manière dont les sons sont utilisés pour réagir aux actions du joueur et provoquer des réactions des NPC est très intéressante et maîtrisée. Chaque "honk" de l'oie, chaque bruit d'objet déplacé est soigneusement calibré pour maximiser l'effet comique et l'impact immédiat de l'action. Ce sens du timing et cette précision sont des éléments que nous cherchons à incorporer pour que chaque perturbation initiée par le joueur ait un effet sonore clair et satisfaisant.

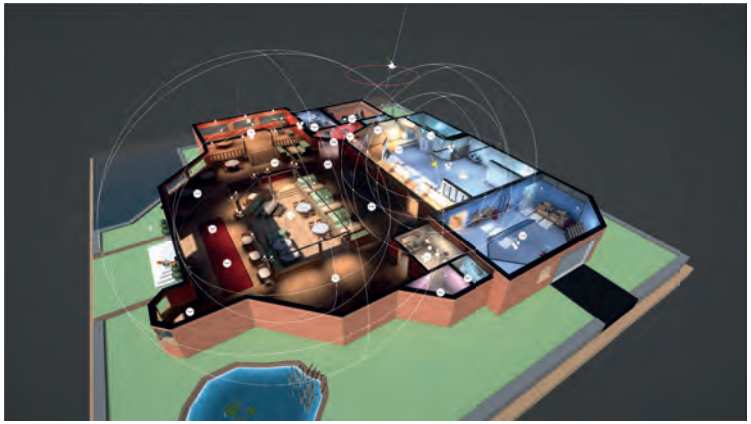
En combinant les meilleures pratiques de ces jeux, nous visons à créer une expérience sonore riche, immersive et amusante qui soutient et amplifie le caractère chaotique et ludique de notre jeu. Les sons ne se contenteront pas d'accompagner l'action, ils en seront une partie intégrante, rendant chaque moment de jeu encore plus engageant et mémorable.

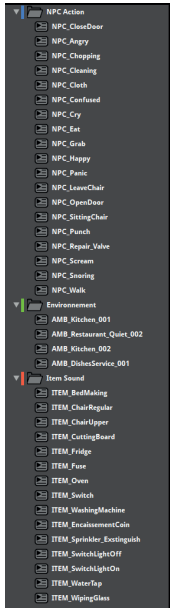
Intégration Fmod

Voici donc comment est organisé notre projet FMOD, avec les 4 principales catégories mentionnées plus haut (environnement, interface, NPC, items).

La majorité des sons présents ici possèdent des multi instruments afin de ne pas créer de répétitivité sonore.





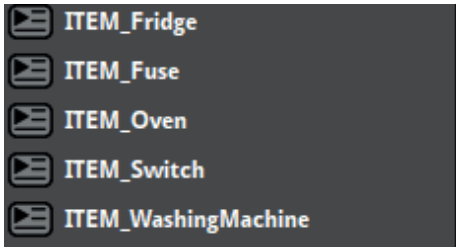


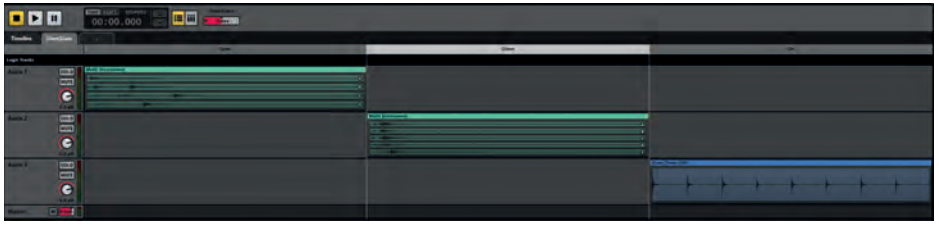
Intégration Fmod

Les quatre sons ci-dessous ont été réalisés avec différents states, afin de rendre le feedback sonore plus intéressants et complexes.

Voici comment nous avons procédé. Chacun de ces events possède 3 states: Open, Close, On. Lorsque l'objet est ouvert par un NPC, l'un des sons du multi instrument du state Open va se jouer. Par la suite, le NPC va fermer la machine et l'un des sons du multi instrument du state Close va se jouer. Durant tout temps que la machine mettra à laver, se jouera le son du state On. Enfin, quand cette dernière sera terminée, le NPC va venir récupérer les affaires, et vont se jouer les sons du state Open, puis du state Close.

Les 3 autres sons fonctionnent de la même manière.





Event List

Voici une partie de notre Event list, qui est organisée de la même manière que notre Fmod, découpé en plusieurs catégories. Chaque son possède plusieurs paramètres. Certains n'ont finalement pas été utilisés, mais nous avons décidé de les laisser dans l'event list.

Comme il est montré dans cette partie de l'event list, la grande majorité de nos sons sont spatialisés, car cela va dépendre de l'emplacement de la caméra dans l'espace de jeu. Plus elle sera éloignée, et plus le joueur ne pourra pas entendre les sons produits par les NPC. À l'inverse, s'il est proche de situations où se déroule des actions, il sera capable d'entendre tous les feedbacks de ces dernières.

La seconde colonne réfère le nom de chaque event Fmod, qui a son équivalent nommé de la même manière dans Fmod. La troisième colonne décrit en quelques mots chaque event, de manière à pouvoir déjà se donner une idée globale.

La quatrième colonne est utilisée pour les références de chaque event. La cinquième colonne sert quant à elle à définir le type de chaque event, et donc la manière dont il sera utilisé. La sixième colonne va quant à elle aider le programmateur, afin de savoir quand il devra déclencher l'event.

La septième colonne a quant à elle été peu utilisée, car la majorité des events sont liés à des animations, et par conséquent elles n'ont pas besoin d'être appelé via code. Les trois prochaines colonnes apportent les dernières informations nécessaires pour les events : leur spatialisation, leur implémentation, et où en est leur production.

Catégorie	event Fmod	Sound description	Ref	Type	trigger condition	Variables	Spatialized	Implémentation	Statut	Commentaires
	NPC_Sad	Sad man who laments	https://www.youtube.com/watch	One Shot	When an action makes an NPC		Oui	Oui	Done	
	NPC_Panic	Panicked man	https://www.youtube.com/watch	One Shot	When an action makes an NPC		Oui	Oui	Done	
	NPC_Confused	Confused man	https://www.youtube.com/watch	One Shot	When an action makes an NPC		Oui	Oui	Done	
	NPC_Happy	Happy man who exclaims	https://www.youtube.com/watch	One Shot	When an action makes an NPC		Oui	Oui	Done	
	NPC_Angry	Angry man	https://www.youtube.com/watch	One Shot	When an action makes an NPC		Oui	Oui	Done	
	NPC_Eat	"Crunch crunch" with	https://www.youtube.com/watch	Loop	When an NPC starts eating		Oui	Oui	Done	
	NPC_CloseDoor	Wooden door who's closed	https://www.youtube.com/watch	One Shot	When a door is closed by NPC		Oui	Oui	Done	
	NPC_Cry	Crying man	https://www.youtube.com/watch	Loop	When an action makes an NPC		Oui	Oui	Done	
	NPC_Coin	Coins ricocheting off	https://www.youtube.com/watch	One Shot	When an NPC client is paying		Oui	Non	Done	
	NPC_Snoring	Snoring man	https://www.youtube.com/watch	Loop	When an NPC is sleeping		Oui	Non	Done	
	NPC_Walk	Man walking on dirt	https://www.youtube.com/watch	Loop	When an NPC starts walking		Oui	Oui	Done	
	NPC_Punch	man punching another with	https://www.youtube.com/watch	One Shot	When an NPC throw a punch		Oui	Non	Done	
NPCAction	NCP_Grab	Cartoon grab	https://www.youtube.com/watch	One Shot	When an NPC grabs		Oui	Oui	Done	

Event List

Voici le QR Code qui redirige vers notre Event List complète.

Mécaniques

Core mechanics

Nos core méchanics vont se découper en deux parties distinctes :

Tout ce qui est lié au mouvement et à la caméra, et ce qui est lié au déplacement d'objets.

En effet, comme expliqué précédemment, le joueur va pouvoir se déplacer sur la map grâce à une caméra contrôlée à la souris. La caméra en mode godmode permet aux joueurs de se déplacer librement sur la carte, offrant une vue d'ensemble du chaos qu'ils créent. Pour accentuer cette liberté, notre sound design s'adapte dynamiquement à la position de la caméra.

Lorsque la caméra se déplace, les sons de l'environnement changent en fonction de la distance et de la direction, créant une sensation de profondeur et d'immersion. Par exemple, en s'approchant d'une scène de chaos, les bruits des objets cassés, les exclamations des NPC et les sons ambiants deviennent plus prononcés, tandis qu'ils s'atténuent en s'éloignant, donnant une impression réaliste de distance et de perspective.

La capacité de la caméra à zoomer et dézoomer permet aux joueurs d'explorer les détails ou de prendre du recul pour voir l'ensemble de la scène. Notre sound design exploite cette fonctionnalité en ajustant l'intensité et la clarté des sons en fonction du niveau de zoom.

En zoomant, les effets sonores deviennent plus précis et détaillés, permettant aux joueurs d'entendre les sons subtils comme le chuchotement des NPC ou le tintement des objets légers. En dézoomant, les sons se fondent progressivement en un paysage sonore plus général, mais toujours évocateur, maintenant l'ambiance globale du jeu tout en offrant une vue d'ensemble.

La possibilité de déplacer et d'attraper des objets est au cœur de notre gameplay. Chaque interaction avec les objets est accompagnée de sons spécifiques qui renforcent la sensation de manipulation et de contrôle. Par exemple, attraper un objet produit un son distinct de préhension. Ces sons sont conçus pour être satisfaisants et immersifs.

Timeline de Gameplay

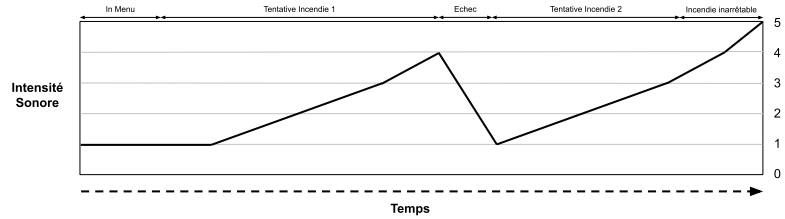
This Was Fine

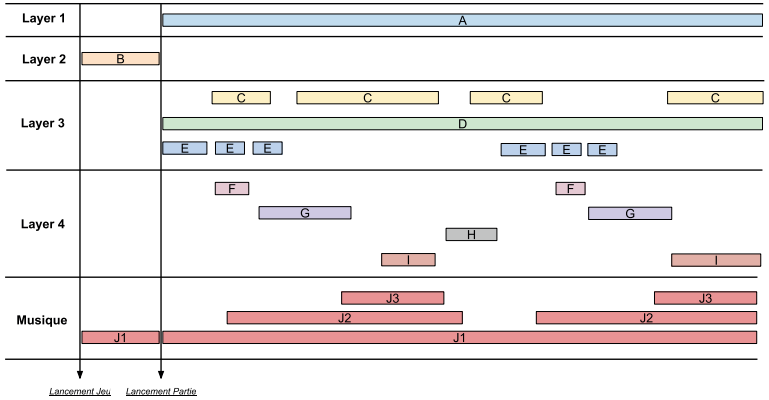
Timeline Sonore d'une partie

Berthier Sébastien
Coustenoble Antoine
Delzangles Paul
Denis Bastien
Helderald Matthias
Limon Théo
Marnet Matthieu

Cette timeline prend pour repère le durée d'une partie moyenne (temps que met le joueur à atteindre un objectif fixé, ici, l'incendie).

On remarque ici trois phases en jeu, la première tentative, l'échec (activation des sprinklers) et la seconde tentative (suivie d'une réussite de la propagation des flammes, condition de victoire).





Layer 1: Environnement	Layer 2: UI	Layer 3: NPCs	Layer 4: Items	Musique
A — Environnement Restaurant Event: - AMB_Kitchen_001 - AMB_Kitchen_002 - AMB_Kitchen_003 - AMB_Kitchen_Knives - AMB_Kitchen_Water - AMB_Restaurant_Quiet_001 - AMB_Restaurant_Quiet_002	B — Layer : UI Event: - UI_StartGame - UI_Menu - UI_Click	C — Emotions NPC Event: - NPC_Sad - NPC_Panic - NPC_Confused - NPC_Happy - NPC_Angry - NPC_Cry - NPC_Scream D — NPC walk: Event: - NPC_Walk (Multi Instru X5) E — NPC Tasks: Event: - NCP_Grab - NPC_Eat - NPC_OpenDoor - NPC_CloseDoor - NPC_SwitchLightOn - NPC_SwitchLightOff - NPC_WaterTap - NPC_WipingGlass	F — Item Cut: Event: - Cutting_Board (Multi Instru X10) G — Oven Feedback: Event: - Oven_Open (Multi Instru x4) - Oven_Close (Multi Instru x5) H — Fire Feedback: Event: - Fire_Event (Multi Instru x4) I — Sprinkler Feedback: Event: - Sprinkler_Event (Multi Instru x4)	J1 — Base track Exemple: - Music Track de base sur laquelle viendra s'ajouter des tracks supplémentaires afin de l'étoffer. J2 — Track Addition 1 Exemple: - Track supplémentaire qui est ajoutée lors de la première phase de réalisation de l'objectif. J3 — Track Addition 2 Exemple: - Track supplémentaire qui est ajoutée lors de la dernière phase de réalisation de l'objectif, là où le chaos est à son paroxysme.

Sound Design

Choix de design

Nos choix de design sonore pour notre jeu ont été guidés par notre désir de créer une expérience immersive et engageante, où chaque action du joueur et réaction des NPC est accentuée par une palette sonore distincte et captivante.

Nous voulons que les joueurs se sentent profondément immergés dans le chaos qu'ils créent. Pour cela, nous avons conçu des effets sonores qui réagissent instantanément aux actions des joueurs. Chaque perturbation, qu'il s'agisse de casser un objet, de déclencher un incendie ou de provoquer une fuite d'eau, est accompagnée de sons distincts et réalistes, rendant chaque action plus tangible et satisfaisante.

Inspirés par des jeux comme Untitled Goose Game et Human Fall Flat, nous avons opté pour un design sonore qui accentue le caractère humoristique et exagéré des situations. Les sons sont légèrement amplifiés et caricaturaux pour renforcer l'aspect comique et décalé du gameplay. Par exemple, le bruit d'un objet qui tombe ou d'un NPC surpris sera volontairement exagéré pour maximiser l'effet comique. Les réactions des NPC sont essentielles pour rendre le monde du jeu vivant et interactif.

Nous avons travaillé sur des effets sonores spécifiques pour chaque type de réaction des NPC, qu'il s'agisse de surprise, de colère ou de panique. Ces sons contribuent à donner du caractère aux NPC et à rendre leurs interactions plus crédibles et engageantes. Pour assurer une expérience de jeu fluide, nous avons mis en place des sons de feedback clairs qui informent les joueurs des résultats de leurs actions. Que ce soit un son confirmant qu'un objectif a été atteint ou un bruit signalant une erreur, ces feedbacks sonores sont conçus pour être intuitifs et immédiatement reconnaissables.

Prise de son

La prise de son pour certains de nos items (Oven washing machine, switch, cutting board, chair) a été un processus enrichissant, mais aussi rempli de défis.

L'un des premiers défis a été de choisir les équipements appropriés pour capturer des sons de haute qualité. Il nous a fallu utiliser des microphones spécifiques pour différents types de sons. De plus, trouver des environnements adaptés pour enregistrer sans bruit de fond indésirable a été crucial mais difficile, nécessitant souvent de se déplacer vers des lieux spécifiques ou de créer des conditions contrôlées.

Capturer le son réel d'objets de notre jeu, tout en s'assurant que ces sons étaient fidèlement représentatifs de l'expérience que nous souhaitions créer, a posé un défi. Parfois, le son réel d'un objet n'était pas aussi percutant ou distinct que nous le voulions pour le jeu, nous avons donc dû équilibrer entre fidélité et impact sonore, en modifiant légèrement les sons en post-production sans perdre leur essence authentique.

Un autre défi a été d'assurer que les sons enregistrés en réel s'intègrent parfaitement avec les actions et animations du jeu. Chaque son devait être soigneusement synchronisé avec les interactions et les événements du gameplay, ce qui a impliqué de nombreux tests et ajustements pour obtenir une correspondance parfaite entre l'audio et le visuel.

Malgré ces défis, l'enregistrement de sons en réel a apporté une dimension de réalisme et de profondeur sonore qui enrichit considérablement l'expérience de jeu.

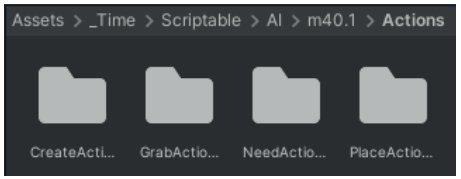
Intégration Unity

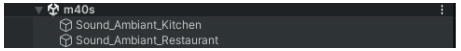
L'intégration sur Unity s'est faite de deux manières : les sons liés aux animations, et les sons liés aux ambiances.

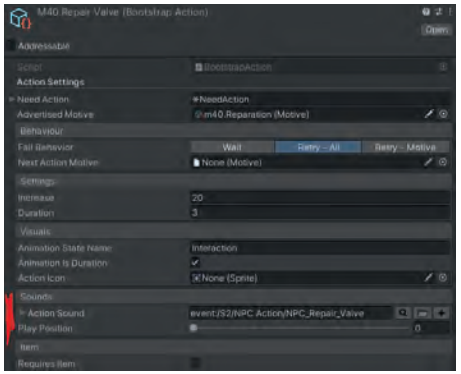
Pour les premiers, comme montrés ci-dessus, chaque animation présente dans le jeu est rangé dans ces dossiers, par type. En les ouvrant, on y trouve les animations, et c'est ici qu'on vient y placer le son lié à l'animation.

Pour les sons liés aux ambiances, ils sont placés à la main directement sur la map.

Certains sons, liés à des VFX, comme par exemple le feu, ont été directement ajoutés au prefab, en leur codant une catégorie dédié à l'ajout d'un son.







Musique

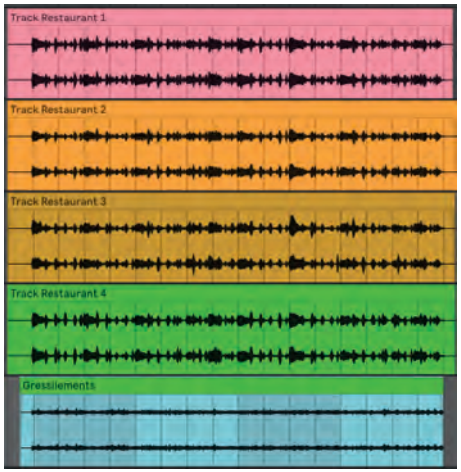
Intentions

This was fine se déroule dans un restaurant et qui dit restaurant dit musique ! Nous avons donc réalisé trois musiques que nous dirons sœurs afin de garnir à la façon de raviolis, les oreilles de nos joueurs.

Pour ce jeu nous voulons proposer aux joueurs une expérience musicale évolutive et en accord avec l'état du système.

Nous voulons également part la musique et l'environnement permettre au joueur de plus facilement identifier l'environnement dans lequel se trouve la bêtise en proposant plusieurs musiques en fonction de l'environnement.

Nous voulons proposer une musique qui évolue vraiment avec le système, nos musiques changent de ton et se modifient en fonction de la montée en puissance de la catastrophe qui occure dans l'environnement.



Process Créatif

Lors de la création de la musique nous sommes passés par plusieurs étapes, il nous fallait savoir ce que nous voulions en terme d'ambiance musicale dans l'environnement, si on faisait une musique qui se séparait ou plusieurs avec chacune leurs différences permettant de mieux discerner quel est l'état de chaque environnement.

Nous avons également dû choisir si la musique devait être constante ou devrait s'effacer et nous avons fini par faire un choix.

Nous avons donc créé une musique de base, nous servant de template, de fondation puis de cette musique en sont sorties trois autres, plus travaillées plus étoffées et avec chacune leur caractéristique rappelant alors la salle de restaurant (très lounge, chill, jazzy), la cuisine (plus énergique, dynamique et usant de percussions plus marquantes) et enfin l'entrepot/les parties techniques. (plus lourd, avec des instruments plus étouffés, des sons des tuyaux...)

Ce procédé nous a permis d'avoir trois musiques avec des points communs tout en ayant chacune leurs petites particularités.

Et les bêtises ?

Pour rendre nos musiques plus interactives, nous avons effectué un travail de recherche notamment pour savoir comment signifier un effet de bêtises. Une bêtise pour nous se devait de déformer la musique, comme si les musiciens étaient entrain de perdre le fil, affectés par la chute de la stabilité de l'environnement.

Nous nous sommes inspiré notamment d'un objet provenant de *Sea Of Thieves,* le coffre aux mille grogs. Ce coffre, si porté par le joueur fait changer la musique et rend le controleur du joueur complètement incontrôlable, donnant l'air d'être très alcoolisé. Cette feature nous a beaucoup inspiré et nous avons essayé de reproduire une sensation relativement similaire avec un VST permettant de déformer le son d'une track.

Pour chacune des musiques, quatre tracks supplémentaires ont été créées et chacune d'entre elles représente un niveau de désordre dans les instruments. Cela nous permet avec Fmod de faire varier l'intensité de « bêtise » dans la musique en fonction de l'état du système de jeu.

Programmation

Vins

Need Based IA

Scene Loading

Dev Console

Organisation

NEED BASED AI

Introduction

Explication

Une Need Based AI en programmation est un principe de sélection d'action pour remplir des besoins. Chaque objet d'un environnement permet de remplir ces besoins, mais de manière conflictuelle. Une action peut diminuer un besoin mais en augmenter un autre. Donc tout les objets d'un environnement s'affrontent pour être sélectionné par un agent.

Le principe central de l'IA est donc un algorithme de sélection pondérée qui va déterminé l'intérêt de chaque action et sélectionner l'action la plus intéressante pour l'IA à l'instant présent.

C'est un principe utilisé dans peu de jeux car très précis. Le plus grand exemple de l'utilisation de ce dernier est Les Sims, une série de jeu visant à simuler la vie de personnage. Mais c'est une simulation imparfaite car, sans l'action du joueur, le système tend à tuer prématurément les Sims. Car ils ne sélectionnent pas forcément les actions nécessaires pour survivre.

Termes

`Need Based AI` : le terme général englobant le principe de l'IA.

`Motive` : une ressource que peut posséder une IA. Elle détermine un besoin, une courbe de scoring et une quantité de ressource à perdre par tick.

`Need Bar` : une jauge de ressource d'un NPC correspondant à un Motive.

`Need Action` : une action réalisable par un NPC via un Advertiser qui fournit une ressource d'un Motive à la fin de celle ci.

`Advertiser` : un objet permettant la réalisation d'une Action.

`Distributor` : un élément gardant une liste d'Advertiser compatible et qui va propager cette liste aux NPCs compatible.

`Area Tags` : des étiquettes attachables aux NPCs, Distributors et Advertisers pour gérer les accès à des Actions en fonction du NPC.

Le système

Structure

Pour faire fonctionner cette IA, tout un système a été mis en place pour faciliter le plus possible la création de situations de jeu par les designer.

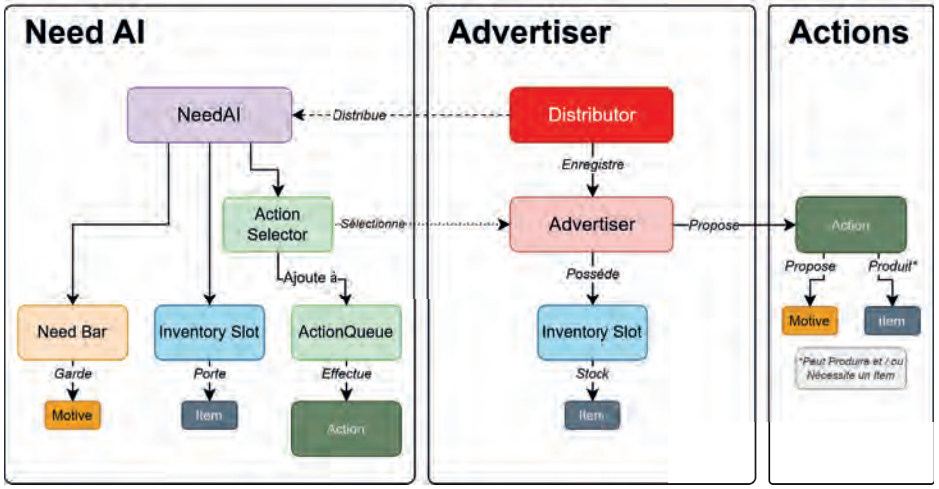
Ainsi, le systèmes est divisé en 3 partie : les IA, les Advertisers et les Actions. Le graphique ci-dessous représente ces 3 systèmes.

La partie IA fait fonctionner les NPC, c'est celle ci qui détermine la meilleure action à effectuer en fonction des quantités de Motive que garde les Need Bars. Elle gère aussi les modifications de ressources de Motive, mais aussi les Items que transporte le NPC. Pour finir elle contrôle l'exécution des actions et des animations.

La partie Advertiser contrôle les différents éléments interactibles de l'environnement. C'est-à-dire que l'advertiser contient une Action et un Inventory Slot. Pour que l'Advertiser soit sélectionné par l'IA, il va être distribué par un Distributor à chaque NPC compatible. De là, l'algorithme de sélection d'action va déterminer la meilleure Action à effectuer qui est proposé par les Advertisers.

Pour ce qui est des Actions, chacune va proposer un Motive. A la fin de celle ci, l'Action va donner à la NeedAI une quantité de ressource au Motive correspondant. De plus, une Action peut nécessiter un Item, ou en produire un.

La structure in-game est un peu plus complexe, mais cette explication représente la majorité du système.

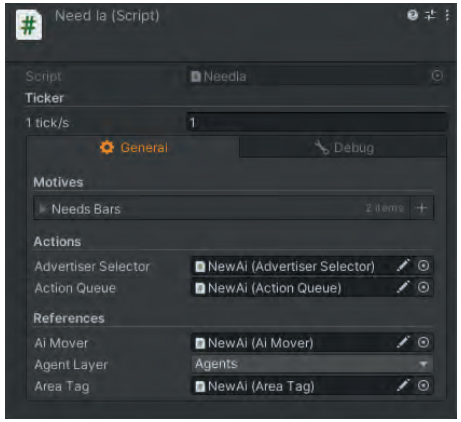


Need AI

Le composant Need AI est au coeur du système, c'est le cerveau de chaque NPC. Il fait le lien entre les différents components qui composent l'IA.

Le component fonctionne sur un principe de tick. C'est-à-dire qu'à chaque tick, il va exécuter une partie de son code. Toutes les 1s, il va déclencher des actions sur les components qui lui sont rattachés.

Ainsi, il va réduire la quantité de ressources de chaque Need Bar, déclencher l'algorithme de sélection d'Action, mettre à jour le pathfinding, etc...



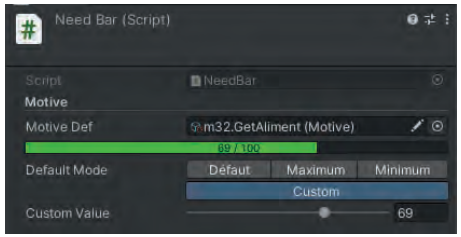
Need Bar

Une Need Bar est un composant qui représente la quantité de ressource d'un Motive pour un NPC.

Une Need AI peut posséder autant de Need Bar que nécessaire. Chacune des bar est rattachée à une Motive Def qui définit le type du Motive qu'elle contient.

Le champ `Default Mode` définit comment la ressource va être initialisé :

- `Défaut` : la ressource est initialisé à la valeur `Default Value` du Motive.
- `Maximum` / `Minimum` : la ressource est initialisé à 100 / 0.
- `Custom` : la ressource est initialisé à la valeur `Custom Value` de la Need Bar.



Action Selector

L'Action Selector est le composant qui gère l'algorithme de sélection des Actions.

Ses 2 paramètres permettent de gérer le tri des Actions.

- `Motive Consideration Range` : le nombre de Motive à prendre en compte.
- `Advertiser Consideration Range` : le nombre d'Advertiser à garder pour le calcul final.

Motive Def

Un Motive est une ressource alant de 0 à 100 qu'une IA peut posséder. Un Motive peut représenter n'importe quoi, aussi bien la faim que le besoin de réaliser une action spécifique.

Pour ce qui est de ses paramètres, il possède une courbe de multiplicateur de score utilisé par l'algorithme, une valeur initiale et une valeur de perte par tick.

Bootstrap Actions

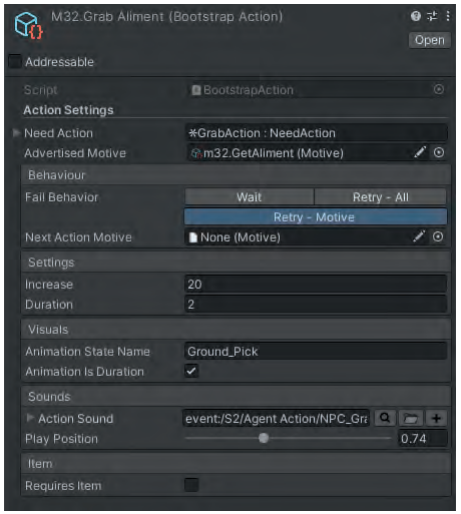
Une Bootstrap Action est un scriptable object qui définit une action référencable par un Advertiser et qui est effectué par un NPC. Une Bootstrap Action ne contient pas réellement de code. C'est une sorte de coquille vide qui permet à un designer de setup des variables depuis l'inspecteur sans toucher au code. Cela est nécessaire à cause de l'architecture utilisée par les Actions.

D'ici, un designer peut définir le type de l'action à effectuer (Need Action, Grab Action, Create Action...), le Motive qui lui correspond, la quantité de ressource obtenue par le NPC, la durée de l'Action, et son comportement en cas d'échec. Voir «Biais de sélection», page 124.

Tandis qu'un artiste peut définir l'animation à utiliser, et le son à jouer et à quel moment.

Un designer peut aussi définir si l'Action requiert un Item ou non. Si oui, il doit fournir une Item Def.

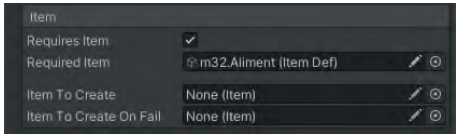
La variable `Increase` correspond à la quantité de ressource que va obtenir le NPC à la fin de l'Action.



Variantes

Il existe des variantes qui possèdent des paramètres différents tel que l'Item que l'Action va créer.

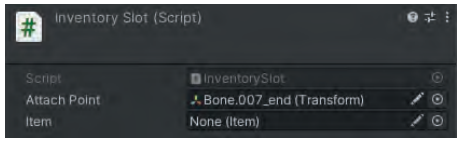
Ci-dessous, un exemple d'une Bootstrap Create Action, elle possède un paramètre pour l'Item à créer avec un succès et un échec.



Inventory Slot

Comme son nom l'indique, l'Inventory Slot est un emplacement d'inventaire. Il peut aussi bien être appliqué à une IA que à un Advertiser.

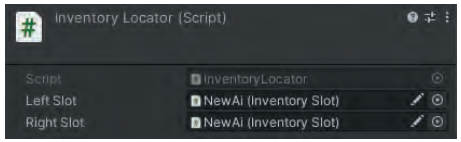
Il a donc pour but d'attacher à un point donné un Item, et à permettre à un NPC d'utiliser cet Item si il est nécessaire pour une Action. Aussi bien si le NPC porte l'Item que si l'Item est placé dans l'Advertiser via l'Inventory Slot.



Inventory Locator

L'Inventory Locator est une extension de l'Inventory Slot pour les NPCs. Effectivement ce composant regroupe des références et des utilitaires par rapport à tout les Inventory Slot d'un NPC.

Ils sont hardcodé pour n'avoir que deux slots car le Game Design ne nécessitait pas plus de slot par NPC (deux mains).



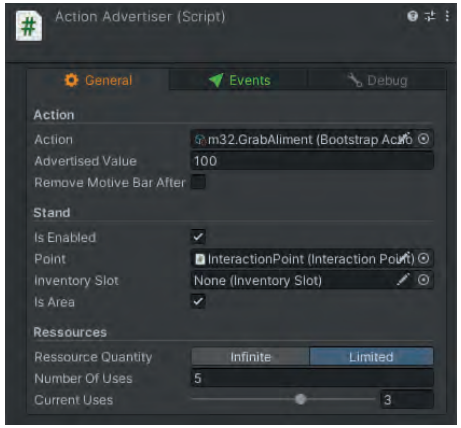
Advertiser

Un Advertiser est un des trois élément essentiel du système. C'est ce composant qui va proposer des actions, mais ce sont les Distributor qui va distribuer aux NPCs ces Advertisers.

Chaque Advertiser possède une `Advertised Value`. Cette variable est la valeur utilisée par l'algorithme de sélection. Elle peut correspondre à la valeur `Increase` de son Action où être totalement différente. Elle permet de manipuler les NPCs en leur faisant croire qu'une Action est très bénéfique alors que la récompense de l'Action Def (`Increase`) est très faible ou inversement. Pour résumer cette valeur est une sorte de fausse publicité pour manipuler les IAs.

En terme de GD, il est aussi possible de définir une quantité d'utilisation limitée à l'Advertiser.

De plus des événements Unity sont disponibles via l'onglet `Events` pour que les designers puissent exécuter des fonctions à des moments clés. La seconde capture d'écran montre ces derniers.

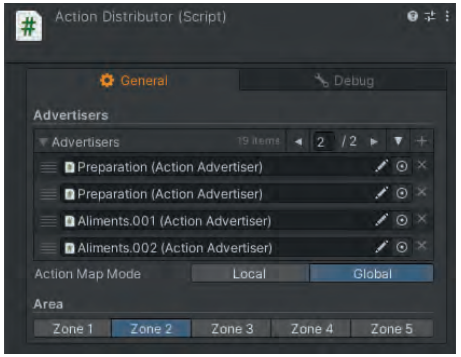


Distributor

Un Distributor est un composant qui stocke des groupes d'Advertiser qui correspondent à ses paramètres.

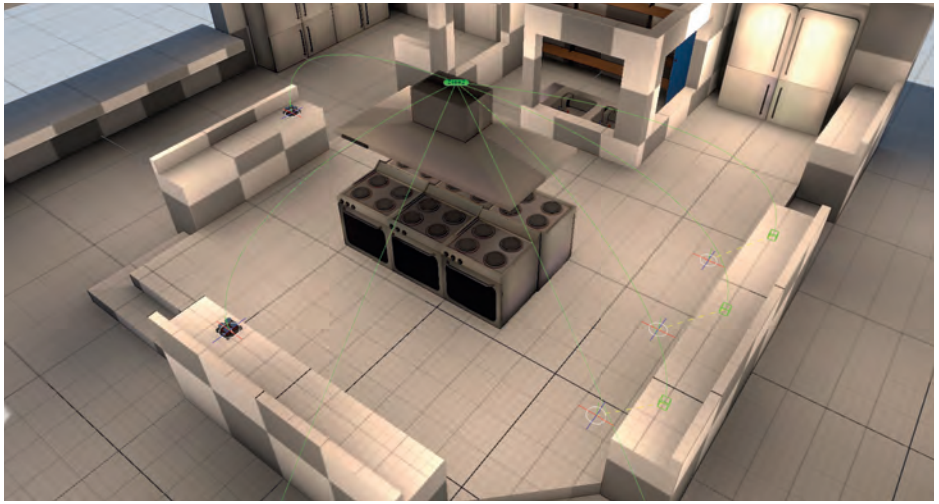
Un Distributor à 2 manières de distribuer ses Advertisers : la 1ère se fait de manière locale via un trigger. La 2nde est appliqué de manière globale via l' «Ai Finder» de la scène.

Chaque Distributor possède un «Area Tags». Via la `Zone` attribué, il récupère les Advertiser correspondant (et seulement ceux qui sont dans son trigger s'il est en mode local), et quand un NPC correspondant à la `Zone` entre dans le trigger (ou à l'initialisation pour le mode global), l'IA reçoit les Advertisers du Distributor. Le NPC prendra donc en compte les Advertisers lié lorsque son algorithme de sélection s'enclenchera.



Gizmos

L'image ci-dessous montre les gizmos du distributor. Dès qu'un Advertiser y est rattaché, une coube partant de l'origine du Distributor est tracé jusqu'à l'Advertiser.



Area Tags

Un Area Tag est un composant optionnel qui permet d'ajouter à un NPC et un Advertiser une `Zone`.

Cela permet aux Distributors de filtrer l'accès à certains Advertisers.

Ai Finder

L'AI Finder est un singleton répertoriant tout les NPCs d'une scène.

Il a pour but d'ajouter et de supprimer des Actions Map aux NPCs compatible avec les Distributor en mode global. Il permet aussi de mettre à jour les Actions Map de toutes les IA lorsque un nouvel Advertiser est créé et ou supprimé.

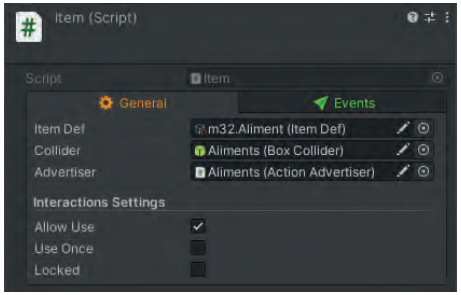
De plus, il permet à la caméra d'alterner son focus entre les différents NPCs de chaque scène.

Item

Un Item représente un objet que les NPCs peuvent porter et utiliser. C'est aussi ce composant qui fait que le joueur peut déplacer un élément.

Chaque Item est lié à un Advertiser qui possède une Action de grab pour que les NPCs puissent prendre l'Item.

Un Item est aussi associé à un Item Def. Ce scriptable object permet aux designers de créer autant de type d'Items qu'ils veulent sans pour autant devoir modifier du code.



Item Def

L'intérêt d'utiliser un Scriptable Object pour représenter une catégorie d'Item. Cela permet aux designers de créer autant d'Item qu'ils le souhaitent.

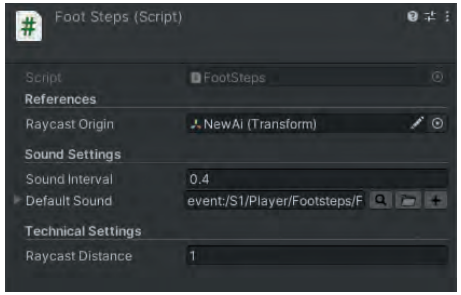
Et côté code, cela permet de comparer les Item Def pour savoir si l'Item que porte le NPC est valide ou non.

Foot Steps

Ce composant à pour but de jouer des sons de bruits de pas lorsque le NPC se déplace. Toutes les 0.4s (correspond à la variable Sound Intervale), si l'IA se déplace, le son de bruit pas est émit.

Le composant possède un système qui détermine le type de la surface sur laquelle se tient le NPC, ce qui permet du côté de FMOD de jouer des sons différents.

Bien sur, si aucune surface spécifique n'est trouvé, le composant fallback à un son par défaut définit par la variable `Default Sound`.



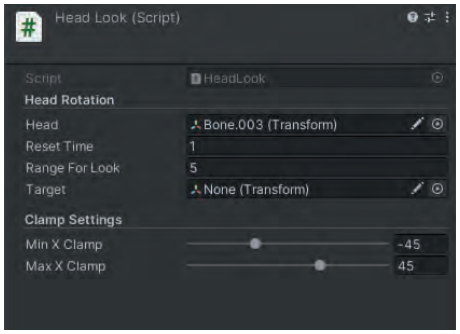
Head Look

Ce composant existe pour ancré les NPCs dans l'environnement de jeu. Il permet à ces derniers de regarder dans la direction vers laquelle est situé l'Advertiser où il se dirigent.

Pour éviter que les NPCs regardent à travers les murs, le script va effectuer un linecast. Si l'objectif est assez proche du NPC et qu'il n'y a pas d'obstacle, il va tourner la tête vers ce dernier.

De plus, le script à aussi pour but de ne pas briser la nuque des NPCs, c'est donc pour cela qu'il y a une limite de rotation.

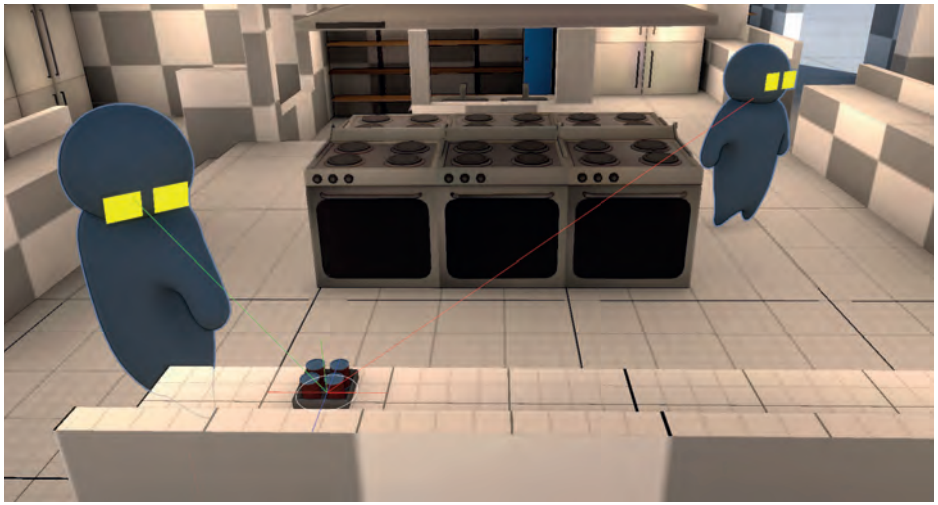
Une rotation sur la durée a aussi été intégré dans le but d'améliorer le feeling de rotation de la tête.



Gizmos

L'image ci-dessous montre les gizmos du Head Look. Ce dernier est vert si le NPC est à portée de regard, sinon il est rouge.

De même que la rotation sur la durée est simulée à l'edit time pour mieux la configurer.



Références Principales

Need Base Ai

Source : http://robert.zubek.net/publications/Needs-based-AI-draft.pdf

Need Based AI est un article rédigé par Robert Zubek pour le journal «Game Programming Gems N°8» publié par Adam Lake le 31 mars 2010.

L'article parle du principe des Need Based AI. En particulier sur la manière dont les Sims (2000) ont redécouvert ce principe. De plus, l'auteur à justement travaillé sur Les Sims, ce qui rend cette article encore plus intéressant.

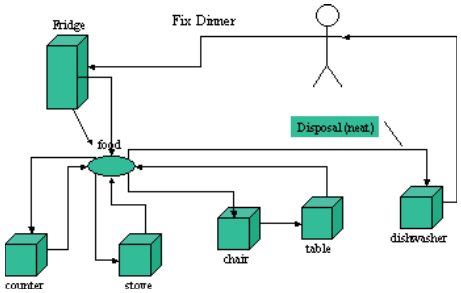
Je me suis servis de cet article pour la base du système, plus précisément la structure et l'algorithme de sélection. Étant donné que notre jeu porte sur ce principe d'autonomie je me suis inspiré des explications de l'article. Tout en réalisant avec ma propre interprétation les systèmes, mais aussi en s'adaptant aux besoins du game design de notre prototype.

L'article est très théorique donc cela m'a permis de mieux comprendre le principe de need based ai, mais aussi à développer mes propres solutions aux différents problèmes de design.

Under the Hood of The Sims

Source : https://users.cs.northwestern.edu/~forbus/c95-gd/lectures/The_Sims_Under_the_Hood_files/v3_document.htm

Under the Hood of The Sims est une présentation pour le cours CS 395 de Game Design du printemps 2002 de la Northwestern University.



La présentation schématise le fonctionnement d'une IA par rapport à l'environnement et ses besoins dans Les Sims.

Le point intéressant de cette présentation est la représentation des chaînes d'actions dans les sims. Étant donné que nous voulions aussi implémenter des chaînes d'actions, cela nous a permis à mieux visualiser comment elles pourraient se réaliser.

L'autre élément pertinent est que la présentation est la représentation visuelle des courbes de besoins et des gains / pertes de ressources proposée par des Advertisers.

The Genius AI Behind The Sims

Source : https://www.youtube.com/watch?v=9gf2MT-IOsg

The Genius AI Behind The Sims est une vidéo faite par Games Maker's Toolkit, et publié en juin 2023.

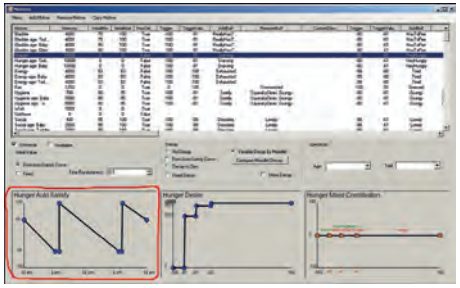
La vidéo est un résumé de 20 minutes sur le Game Design des sims allant à la surface du fonctionnement des différents systèmes qui font que les IA soient des IA autonomes.

Elle a été très intéressante en terme de design car elle nous a expliqué d'un point de vue Game Design le fonctionnement du principe du need based ai. Nous nous sommes donc principalement basé sur le principe des courbes de besoins et des advertisers.

Modeling Individuals Personalities in The Sims

Source : https://www.gdcvault.com/play/1012450/Modeling-Individual-Personalities-in-The

Modeling Individuals Personalities in The Sims est un talk de la GDC 2010. Elle a été faite par Evans Richard.



La présentation entre en profondeur dans les systèmes de comportements, des systèmes de personnalités et de relations des sims. Bien que très intéressante, seule la partie sur les comportements fut réellement utilisé car le reste était en dehors du scope de notre game design.

Les slides 15 à 37 furent donc les plus intéressantes. En particuliers celles contenant du pseudo code. En effet, ces dernières sont à propos des problème d'optimisations rencontrés dans les jeux précédents et ce qu'il avait fait pour améliorer le code des Sims 3.

Par exemple, la manière dont il explique en slide 21 et 22 que chaque besoin est attaché à une liste d'action pour diminuer les itérations nous a inspiré. Effectivement, notre système de jeu génère ce que nous appellons une Action Map, une classe qui contient une association entre un besoin et ses Advertisers liés. Ce qui permet de seulement itérer à travers les advertisers concernés.

Algorithme de sélection

Overview

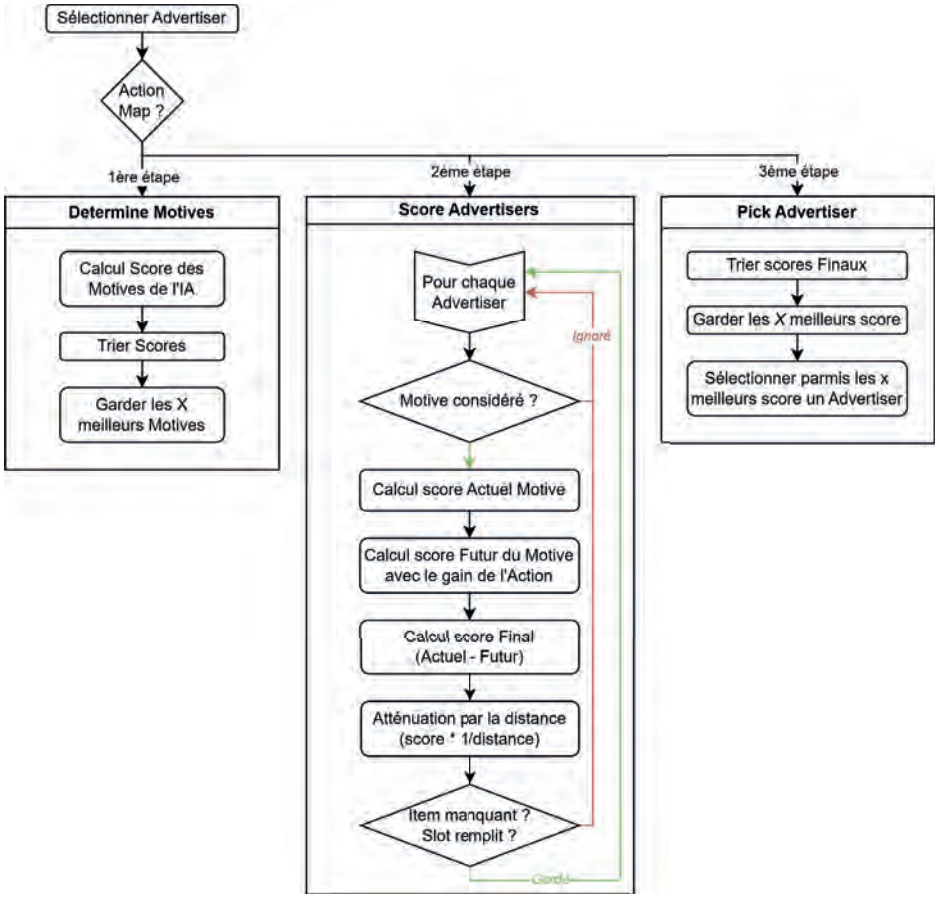
Chaque NPC possède un Action Selector, ce component est celui qui va calculer le score de chaque Advertiser en fonction des Need Bars de l'IA. En fonction du contexte environnemental, l'IA va connaître les interactions possibles pour chaque Motive qu'elle possède.

La sélection de l'Advertiser à utilisé se fait en plusieurs étapes. Le graphique ci-dessous montre le processus de calcul du score d'une action en pseudo code.

Le processus peut être séparé en 3 étapes. La 1ère correspond à la sélection des Motives que le système va prendre en compte pour la suite de l'algorithme.

La 2de partie, va calculer le score de chaque Advertiser proposant une Action ayant un Motive correspondant à l'un des Motives sélectionné précédemment.

La 3ème étape est la sélection de l'Advertiser grâce à un calcul aléatoire pondéré.



Pré-Algorithme

Avant même que l'algorithme se déclenche, des vérifications sont mis en place pour éviter de l'exécuter trop souvent. En effet, ce dernier est plutôt intensif donc plutôt que de l'exécuter à chaque update, il s'exécute sur un principe de tick. Mais pas non plus à chaque tick.

Dès que l'Action Queue du NPC possède au moins une Action, l'algorithme ne peut pas s'exécuter car il n'y a pas besoin de chercher une Action à effectuer alors que le NPC en a déjà une.

De plus, l'algorithme peut être déclenché automatiquement par l'IA lorsqu'elle en a besoin immédiatement. Voir «Biais de sélection», page 124.

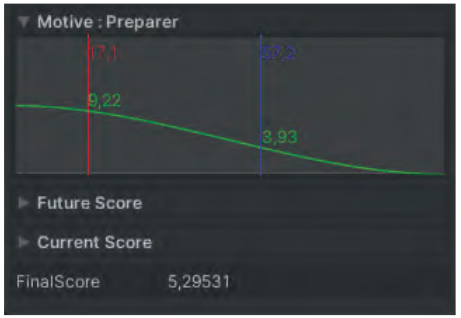
Détermination des Motives

Depuis la liste des Need Bars, l'Action Selector va calculer le score de chaque Motive en fonction de la valeur actuelle de la Need Bar et de la courbe de multiplicateur de score du Motive.

Il va ensuite trier les Motives par score croissant. Finalement, il ne gardera que les Motives compris entre le premier Motive et le `Motive Consideration Range`.

L'image ci-dessous représente la courbe de multiplicateur d'un Motive (vert), le score actuel (rouge), le score futur (bleu), et les multiplicateurs au score actuel et futur (vert).

Dans cette étape, l'algorithme ne prend en compte que le score actuel.



Scoring des Interactions

Ensuite, le système va itérer à travers l'Action Map du NPC. Pour chaque Motive de la map, il va regarder si le Motive est compris dans ceux sélectionnés précédemment. Si oui, il va pour chaque Advertiser associé à ce Motive dans la map, calculer le score final de l'Action. Il est calculé en 5 étapes.

1. L'algorithme calcule la valeur future de la Need Bar. `FuturScore = NeedBar.Value + Advertiser.AdvertisedValue`. Cela correspond à la valeur bleue du graphique.

2. Le multiplicateur de score actuel est calculé. Il correspond à la valeur de la courbe de multiplicateur de score du Motive à la position `Δt = NeedBar.Value`. C'est donc le première valeur verte.

3. Même chose qu'à l'étape précédente. Sauf que `Δt = FuturScore`. C'est donc la seconde valeur verte du graphique.

4. Le score final est obtenu en soustrayant le score actuel du score futur. `Score = MultiplicateurFutur - MultiplicateurActuel`.

5. La distance entre le NPC et l'Advertiser est calculé, puis son inverse est calculé. Pour finir, le score est multiplié par l'inverse. `Score = Score * (1 / distance)`.

De là, si l'Action nécessite un Item et que le NPC ne possède pas ce dernier, ou que l'Inventory Slot de l'Advertiser ne possède pas l'Item, l'Advertiser est ignoré.

Sinon, l'Advertiser et son score est ajouté à une liste d'Advertiser possible à utiliser.

Sélection de l'Advertiser

Après avoir calculé tout les scores, l'Action Selector va sélectionner l'action à effectuer parmis les actions ayant le plus grand score. Le nombre d'interactions sélectionnées par le choix aléatoire est définit par `Advertiser Consideration Range`.

Par exemple : il y a 20 Advertisers de sélectionnés et la variable est de 5. Le système va sélectionner parmi les 5 Advertisers ayant le plus grand score l'Action à effectuer.

Une fois les Advertisers finaux sélectionnés, une sélection procédurale est effectuée. Tout d'abord, la somme des scores est calculée, et avec cette somme, une valeur aléatoire entre 0 et le résultat est sélectionnée.

De là, le système compare la valeur aléatoire au score des Advertiser. Il sélectionne l'Advertiser où la valeur est située entre le score de l'Advertiser et le score de l'Advertiser suivant.

Déclenchement de l'Action

Une fois l'Advertiser sélectionné, le Need AI va se diriger vers le point d'interaction de l'Advertiser si il en est trop éloigné. Il fait cela avec une Action de type `Move Action`. Une fois qu'il est à porté du point d'interaction, le NPC effectue l'Action de l'Advertiser.

A la fin de l'Action, la Need Bar correspondant au Motive de l'Action va être en mise à jour en recevant une valeur définie dans une Bootstrap Action. Voir «Bootstrap Actions», page 115.

Biais de sélection

Dans certaines situations, l'algorithme peut être déclenché automatiquement par le NPC sans passer par les ticks. Dans le cas où l'Advertiser de la prochaine Action de l'IA est déjà utilisé par un NPC, l'algorithme est relancé, possiblement avec un biais.

De même que certaines Actions peut demander une exécution de l'algorithme pour effectuer une Action correspondant à un certains Motive.

Il existe deux type de biais pour le déclenchement de l'algorithme.

1. Retry - Motive : ce biais fait que l'algorithme va sauter la 1ère étape pour obtenir seulement les Advertiser utilisant ce type de Motive. Ce biais permet donc d'effectuer une Action d'un Motive en particulier sans pour autant forcer un Advertiser en particulier.

2. Ignore : Dès qu'un Advertiser est sélectionné par le NPC, il va être ignoré lors de la prochaine exécution de l'algorithme. Cela à pour but de faire que les IA aillent à d'autres Advertiser plutôt que de toujours aller au même.

Détail d'implémentation : Actions

Structure

Étant donné que le projet allait nécessiter un certains nombre d'Actions, où chacune d'entre elles possèdent des paramètres différents, nous avons opté pour une structure orienté données.

Elle a pour objectif de séparer le code des Actions des données des Actions. Pour cela, les Actions existent de deux manière dans le projet : les Bootstrap Actions et les Need Actions.

Les Bootstrap Actions sont des scriptable objects modifiable depuis Unity, ils contiennent les paramètres. Tandis qu'une Need Action est un script «C# pur» comprenant le code du fonctionnement de l'Action.

Le diagramme ci-dessous représente cette structure. Du côté Editor Unity il y a les Bootstrap Actions et ses variantes venant de classes héritées. De l'autre, l'arbre d'héritage des différents types d'Action. Seule les Need Action (en vert), peuvent être utilisé par les designers. Les autres étant des Actions s'exécutant dans des situations précises ne dépendant pas du système de besoin des NPCs.

Bootstrap Action

Voir «Bootstrap Actions», page 115 pour les détails de variables.

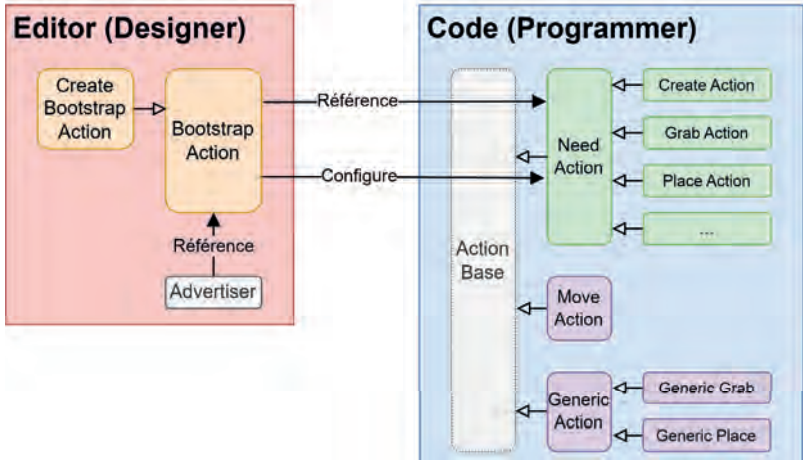
Un BootStrap Action est un scriptable object permettant aux designers de définir une Need Action configurable et assignable à un Advertiser.

Ce sont des scriptable objects, car cela permet une réutilisation de ces derniers. Ils font le lien entre la partie designer et programmer : les données de configuration sont séparées du fonctionnement de l'Action.

C# Pur

Comme dit précédemment, Action Base et ses héritières sont des classes C# pure. Elles contiennent le code faisant fonctionner les Actions.

Utiliser des classes C# pure à l'instar des MonoBehaviour est un choix de design. Les Actions sont souvent effectuées, par des NPCs différents, ainsi des MonoBehaviour seraient souvent instanciés et détruit. Pour éviter cela, nous avons opté pour des classes C# pure, qui sont beaucoup moins lourdes.

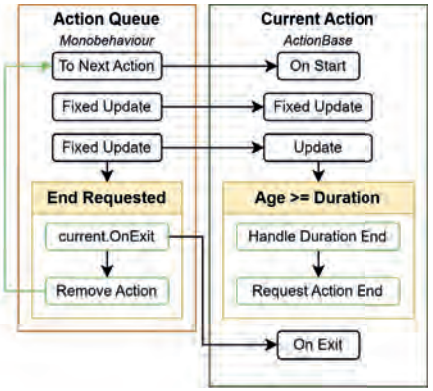


State Machine

Au début nous ne savions pas trop que faire pour l'exécution des Actions. À ce moment, nous avions décidé que une Action était ce que faisait un NPC à un stand.

Mais après quelques itérations, nous avons constaté que associer tout ce que faisait un NPC à une Action permettait de gérer tout les comportements des NPCs de la même manière.

De là est né la structure des Action Bases et avec l'idée de faire une State Machine. Car chaque Action était devenu un état qui définissait ce que faisait le NPC. C'est ainsi que l'Action Queue a été créée car il fallait un moyen pour un NPC de posséder une chaîne d'actions dynamique. En effet, si le NPC doit faire l'action de déplacement pour atteindre l'Action Advertiser sélectionné, il doit garder en mémoire l'action de déplacement, et l'action de l'advertiser.



Instantiation

Ainsi, l'Action Queue gère l'état du NPC, mais aussi les transitions entre les différents états. Les ActionBases étants des classes C# pure, elles sont instantiés lorsqu'elles sont ajouté à la queue, et détruite lorsqu'elles en sont retirés.

L'objet qu'obtient la queue est une BootstrapAction, non une ActionBase. Pour obtenir une instance de l'ActionBase définit par la BootstrapAction nous avons dût instantier une classe depuis une référence à une instance d'un type.

Pour ce faire nous avons donc écrit ces quelques lignes à l'intérieur de BoostrapAction. Avec cette manière d'instantiation il est impossible d'appeler le constructeur avec des arguments. Nous appelons donc une méthode qui setup les variables de l'action.

Nous passons comme argument la BootstrapAction, de là, l'ActionBase va récupérer les variables de la BootstrapAction pour utiliser les valeurs définies par les designers.

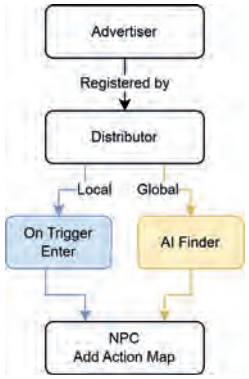
State Machine

Le diagramme suivant montre le fonctionnement de notre state machine et du rapport entre les Actions et la queue.

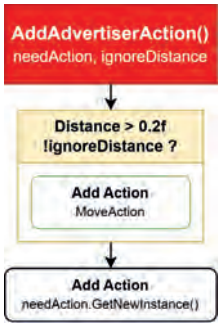
```
public NeedAction GetAction(ActionAdvertiser advertiser)
{
    Assert.IsNotNull(NeedAction, message: $"{name}'s NeedAction is null.");

    var state = Activator.CreateInstance(NeedAction.GetType()) as NeedAction;
    state?.ManualSetup( bootstrapper: this, advertiser);
    return state;
}
```


Lifecycle d'une Action

À l'edit time, chaque Advertiser est assigné à un ou plusieurs Distributors. C'est au lancement de la scène que les Advertisers vont être distribué aux NPCs. Soit automatiquement par les Distributors globaux, soit par trigger par les Distributor locaux. De là, les NPCs connaissent l'Advertiser et il pourra être considéré par l'algorithme de sélection.

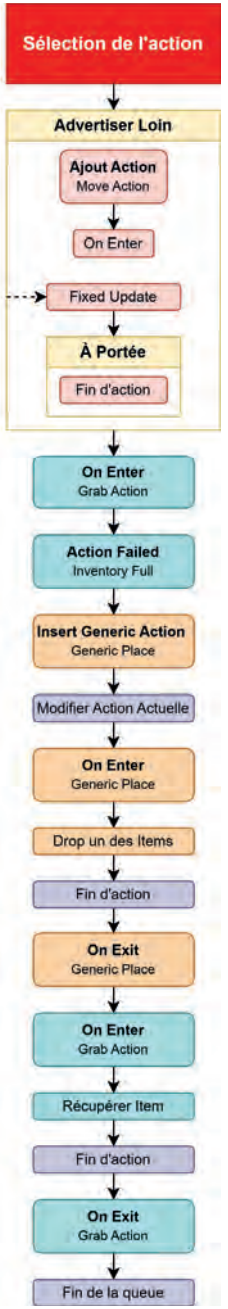


Une fois qu'une NeedAction est ajoutée à la queue, elle va rester dormante jusqu'à ce qu'elle soit utilisée par la queue.



À ce moment là, elle va devenir l'état actuel de la state machine. Et sera contrôlée comme indiquée par le diagramme de la page précédente.

Exemple d'une chaine d'action



SCENE LOADING

Chargements

Introduction

À l'origine, notre premier prototype était composé de plusieurs niveaux séparés en scène unity. Il nous fallait donc un moyen de transition propre entre les différents niveaux, et bien-sur, le menu principal.

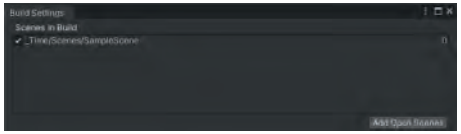
Ainsi, nous avons développé un système de chargement de scène comportant un écran de chargement via le package des Addressables d'Unity.

Addressables vs Build Order

Dans Unity, il y a toujours eu une seule manière de charger des scènes in-game depuis un build. Pour cela il fallait passer par le Build Order et avoir des références faibles par des numéros de scènes ou des noms de scènes. Mais actuellement, si l'on regarde notre Build Order ci-dessous, il est vide.

Cela est possible grâce à l'utilisation du package officiel des Addressables. Un système puissant permettant de charger de manière asynchrone des assets et des scènes.

En comparent le code pour charger une scène de manière async, ce n'est pas si différent. Sauf que plutôt d'utiliser un nom ou un index, les Addressables utilisent l'adresse de la scène.



L'intérêt des Addressables est de pouvoir charger un asset depuis une addresse différente du nom de l'objet, ainsi les références ne sont pas perdues si un asset est renommé. En plus, le système permet de créer des asset bundles, ce qui permet au moteur de ne charger que les assets nécessaires au bon moment.

L'image ci-dessous montre le groupe «Scenes» qui contient les addresses (colonne 1) des scènes pour les référencer depuis le code.

C'est beaucoup moins contraignant que le Build Order car il n'y a pas d'ordre à respecter.

Flow de chargement

La boîte rouge est le point d'entré du chargement de carte. Les boîtes jaunes des conditions à vérifier pour exécuter le contenu à l'intérieur. Une ligne rouge signifie un échec de la vérification, tandis qu'une verte est une réussite. Le bleu correspond au lancement d'une coroutine.

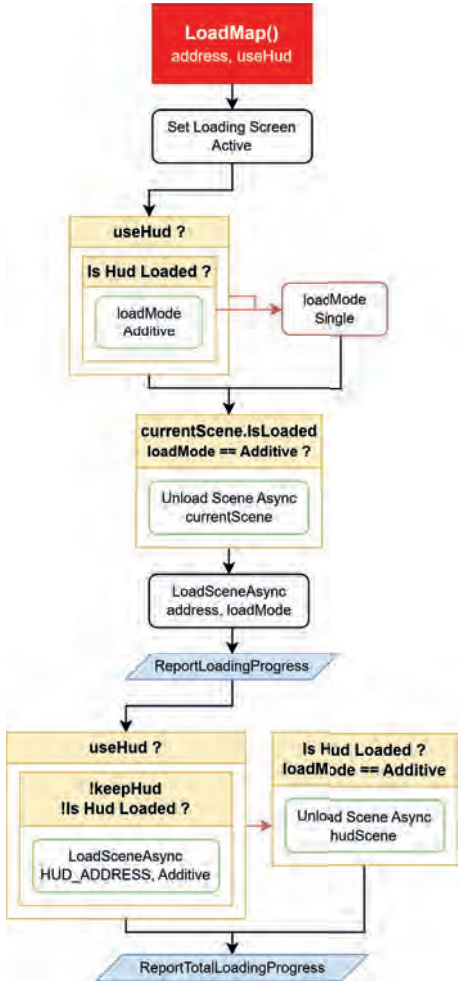
Lorsque le système demande le chargement d'une scène valide, l'écran de chargement s'active. Ensuite, s'il faut charger le hud, il regarde s'il est déjà chargé. Si oui, il définit le mode de chargement en additif pour éviter de remplacer le hud.

De là, si une scène est déjà chargée et que le mode de chargement est en additif, il va décharger la scène chargée. Car si l'on décharge la scène alors qu'aucune autre scène n'est chargée, et que l'on est en mode single, il y a une erreur car plus aucune scène n'est chargée.

Ensuite, on assigne à une AsyncOperation le chargement de la scène principale avec le mode définit précédemment. Juste après une coroutine est lancée, elle permet de set la scène chargée en tant que scène active à la fin du chargement.

Après, si l'on veut utiliser le hud, on vérifie qu'il est bien chargé, et si il l'est, on empêche le rechargement de ce dernier. Sinon, on l'ajoute à l'AsyncOperation. Si on ne veut pas le HUD, on vérifie qu'il est chargé et que le mode de chargement est additif pour le décharger manuellement. En effet, si le mode de chargement était en single, il aurait été déchargé automatiquement par unity.

Finalement, on exécute une seconde coroutine qui désactive l'écran de chargement quand l'AsyncOperation se termine.



Code

Voici les lignes de code pour charger une scène en mode async, tout en récupérant une AsyncOperation. Via le Build Order et les Addressables.

```
// SceneManager (Async)
var loadMapOperation = SceneManager.LoadSceneAsync(0, loadMode);

// Addressable (Async)
var loadMapOperationHandle = Addressables.LoadSceneAsync(mapAddress, loadMode);
```

Le bonus de ce système

Les scènes du jeu ne sont pas connu au chargement du jeu, car une seule scène est présente dans le build order. Ainsi, nous avons du développer un moyen de trouver toutes les scènes existantes dans les assets bundles du projets.

Ce système va donc regarder à travers l'asset bundle des scènes pour trouver et créer une liste d'adresse de scène.

Il est plutôt simple : pour tout les assets trouvés dans le bundle, il vérifie que l'asset est une SceneInstance. Et ci c'est le cas, il récupère le nom de la scène et l'ajoute à la liste. Une fois la tâche finie, la liste est assignée et le système dit qu'il a terminé de charger les scènes.

Ce bout de code là nous permet donc de trouver au runtime toutes les scènes du projet sans devoir refaire un build complet. Il suffit juste de mettre à jour le build des addressables. Nous avons donc une manière d'ajouter des scènes dans le projet sans refaire de build, tant qu'elles n'ajoutent pas de nouveaux scripts.

Optimisations

Étant donné que cette opération s'exécutait au tout début du jeu, elle causait un lag de première frame. Pour réduire ce lag, nous avons transformé la méthode en Task C#.

Via l'utilisation du keyword await, cela permet d'appeler la méthode de manière asynchrone, sans pour autant faire du multithreading. Avec ceci nous avons observé une réduction de ce lag de première frame.

```
private async void SetMapList(AsyncOperationHandle operation)
{
    IList<IResourceLocation> locations = operation.Result as IList<IResourceLocation>;
    MapAddressList = await GenerateMapList(locations);
    MapListLoaded = true;
}

1 usage  TjgL
private static Task<List<string>> GenerateMapList(IList<IResourceLocation> locations)
{
    var mapList = new List<string>();
    foreach (IResourceLocation location in locations)
    {
        if (location.ResourceType != typeof(SceneInstance))
            continue;

        if (location.InternalId == HUD_ADDRESS)
            continue;

        string name = System.IO.Path.GetFileNameWithoutExtension(location.InternalId);
        mapList.Add(name);
        Debug.Log(message: $"MapHandler::LoadMapList::Found map {name} at address: {location.InternalId}");
    }

    return Task.FromResult(mapList);
}
```

Workflow multi-scene

Qu'est ce que c'est ?

Travailler avec un workflow multi-scene est une manière de séparer différents éléments d'une scène en plusieurs sous-scènes.

Pour notre projet nous avons opté d'avoir 2 à 3 sous-scènes par scène.

Nous avons donc la scène principale contenant tout les assets graphiques.

Une scène de lighting contenant, les lumières, les lights probes et les reflections probes.

Ensuite, nous avons une scène tech contenant tout les gameobjects à script tel que les Advertisers et Distributors.

Et finalement, nous avons une scène audio pour les sons d'ambiance.

De plus, nous avons une scène de HUD contenant le menu de pause.

Avantages & Inconvénients

L'intérêt de ce workflow est pour le travail en collaboration. Effectivement, malgré le fait que les fichiers de scène unity peuvent être merge, il y a souvent des problèmes à cause de modification incompatible entre deux versions.

Mais en séparant la scène en sous-scènes, l'artiste peut faire l'environnement sans pour autant remplacer le travail du game designer lors d'un push. Étant donné que les scènes sont différentes, tout le monde peut travailler de son côté sans ruiner le travail d'un autre, car personne ne travaille sur la même scène.

Le bémol de ce principe est qu'il y a donc beaucoup plus de scène existante, que dans l'éditeur il faut les charger manuellement pour voir la scène complète. Et in-game, il faut que le système de chargement de scène supporte ce principe.

Le premier problème a été réglé via un tool. Tandis que le second n'en est pas un car, le système de chargement du prototype a été pensé pour charger plusieurs scène dynamiquement : la scène de jeu et la scène de hud. Ainsi, il a juste fallu adapter quelques lignes pour adapter le chargement.

Fonctionnement

Comme expliqué précédemment, notre système de chargement de scène était déjà capable de charger plusieurs scènes. Il nous a juste fallu faire quelques modifications pour s'adapter aux modification prévues.

Nous avons créé une méthode locale dans celle de chargement de scène pour charger des scènes automatiquement si elles correspondent à un nom donné.

Chaque scène de notre projet est appelé de cette manière : `mXX_Nom`. Le code ci-dessous va donc chercher si une scène existe avec le nom `mXX[lettre]`, si oui il va l'ajouter à l'AsyncOperation de chargement.

```
void LoadAdditionalScene(char letter)
{
    var sceneName string = mapAddress.Split(separator: '_')[0] + letter;
    try
    {
        var sceneAddress string = _mapHandler.GetMapAddress(sceneName);
        var loadOperation = Addressables.LoadSceneAsync(sceneAddress, LoadSceneMode.Additive);
        _loadHandle.Add(loadOperation);
    }
    catch (KeyNotFoundException)
    {
        Debug.LogWarning(message: $"Impossible de charger la scène additionnelle : {sceneName}.\n" +
            $"Ignorez cette erreur si la scène n'a pas de scène '{letter}' associée.");
    }
}
```

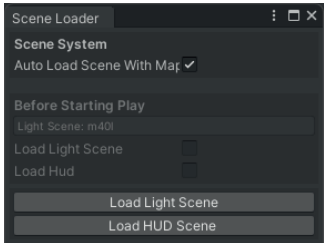
Tooling

Pour faciliter le workflow multi-scene, nous avons développé un outils qui s'accroche au système de lancement du play mode d'unity.

Au début il permettait juste de recharger la carte via le système de chargement de scène, comme si l'on était en build. Cette option permettait aussi de charger le HUD automatiquement.

Ensuite, lorsque ce nouveau workflow fut adopté, l'outil a été étendue avec une EditorWindow.

La mise à jour de l'outil à ajouter de quoi charger automatiquement la scène de lumière et de hud, sans passer par l'option de rechargement automatique. Et elle a aussi ajoutée des boutons pour charger dans l'éditeur la scène de lumière de la scène ouverte et le HUD.



Infos

C'est une classe en plus de la fenêtre qui s'occupe de charger les scènes.

Elle se sert de l'event `playModeStateChanged` de `EditorApplication` pour charger les scènes au bons moments.

Fonctionnement

Attributs

En C#, un attribut est un élément que l'on met devant une déclaration pour effectuer quelque chose.

```
[Command("npc_clearall", "Clear all action queues of all NPCs.")]
```

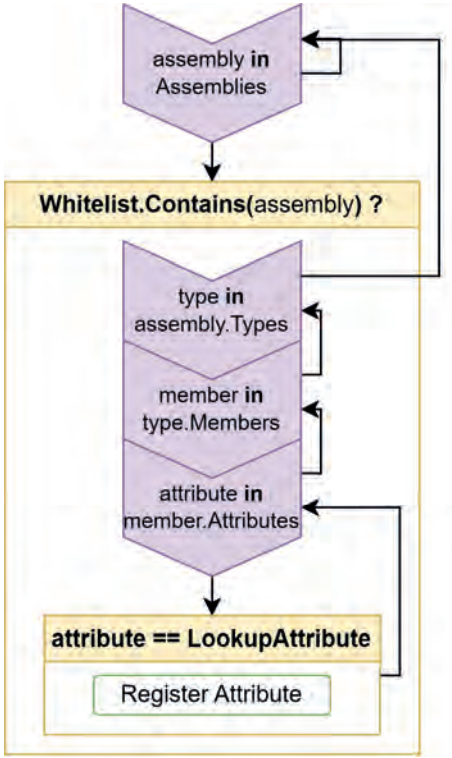
Pour faciliter l'ajout de commande nous avons décidé d'utiliser des attributs pour marquer une méthode comme une commande. L'image ci-dessus en est un exemple. Il suffit juste d'ajouter cet attribut à n'importe quelle méthode et de le paramétrer pour transformer une méthode en une commande.

Mais utiliser des attributs pour cela rend la tâche d'implémentation beaucoup plus difficile car il faut donc trouver tout les attributs au runtime et ensuite créer de quoi appeler la méthode liée.

Reflections C#

Il n'existe qu'une seule manière d'obtenir tout les éléments suivants un pattern en C#, et cela passe par les reflections. Nous cherchons donc comme pattern une méthode static et public ou static qui contient cet attribut.

Le diagramme suivant montre le fonctionnement de la recherche via les reflections. Il y a beaucoup d'itérations mais on ne peut pas y faire grand choses. Pour réduire ces itérations, nous utilisons une whitelist qui permet de chercher les attributs seulement dans quelques assemblies.



Diagramme

Les flèches violettes représentent des boucles foreach.

Le diagramme contient évidemment du pseudo code car il n'y a pas assez de place pour écrire les lignes complètes des reflections.

Création de la commande

Une fois toutes les commandes trouvées, nous créons pour chaque commande une entrée dans un dictionnaire. Avec comme key le nom de la commande et comme value un objet commande contenant un delegate pour la méthode à appeler.

Utiliser des delegates permet de cache la commande au lancement. Mais cela vient avec un inconvénient : toutes les méthodes doivent donc avoir la même signature. C'est à dire qu'elles doivent tous avoir les mêmes parameters.

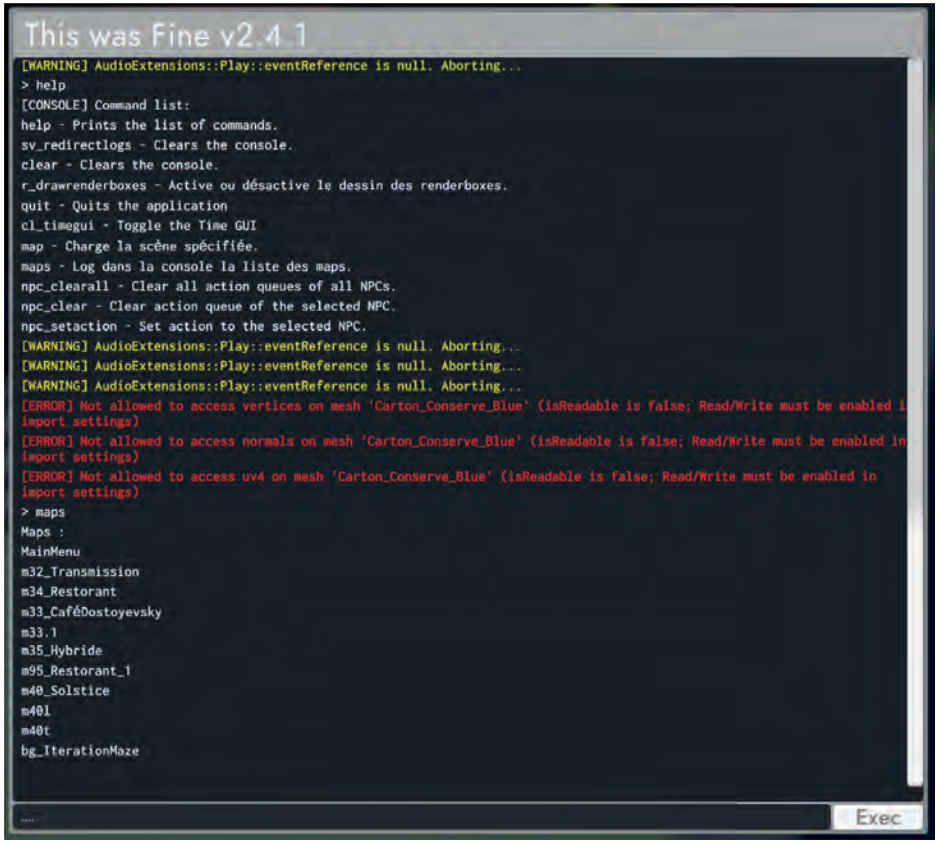
En effet, le C# est un langage static, il doit donc connaître les types de chaque variables au compile time.

Parsing des inputs

Pour réduire le problème cité précédemment, nous avons créé une struct qui va recevoir tout les arguments de la commande lorsque l'on va l'écrire dans la console. De là, elle va parse les arguments pour que la méthode de la commande puisse s'en servir.

Le parsing d'une input de console est simple. Le système vérifie que le nom de la commande existe. Et, si oui, il va séparer la commande en plusieurs parties délimités par des espaces. Tout les mots après le premier sont considérés comme des arguments.

Finalement, la méthode est appelée via le delegate. Avec comme argument la struct, créée depuis les arguments de l'input.



La console

Les commandes

Voici une liste des commandes de notre prototype, les - correspondent à du vide.

Commande	*Description*	*Paramètres*	*Valeurs Acceptées*
npc_clearall	Clear la chaîne d'action de tout les npcs.	-	-
npc_clear	Clear la chaîne d'action d'un npc.	string	<Nom Npc>
npc_setaction	Définit l'action d'un npc.	string, string	<Nom Npc>, <Adresse Action>
map	Charge la scène spécifiée.	string	<Adresse Scène>
maps	Imprime la liste des adresses de scènes dans la console.	-	-
help	Imprime la liste de commandes dans la console.	-	-
clear	Vide le contenu de la console.	-	-
sv_redirectlogs	Redirige les logs Unity vers la console.	bool	<false / true>
r_drawrenderboxes	Active ou désactive l'affichage des renderboxes.	int	<0 / 1>
quit	Ferme le jeu / quitte le play mode.	-	-
cl_timegui	Affiche l'interface de contrôle du soleil.	bool?	<- / false / true>

Raccourcis

La console possède des raccourcis claviers pour faciliter l'utilisation. Pour ouvrir et fermer la console il suffit d'appuyer sur ².

Pour exécuter une commande, il faut appuyer sur enter. Il est possible de naviguer entre les différentes commandes utilisée via les flèches directionnelles haut et bas.

ORGANISATION

QA

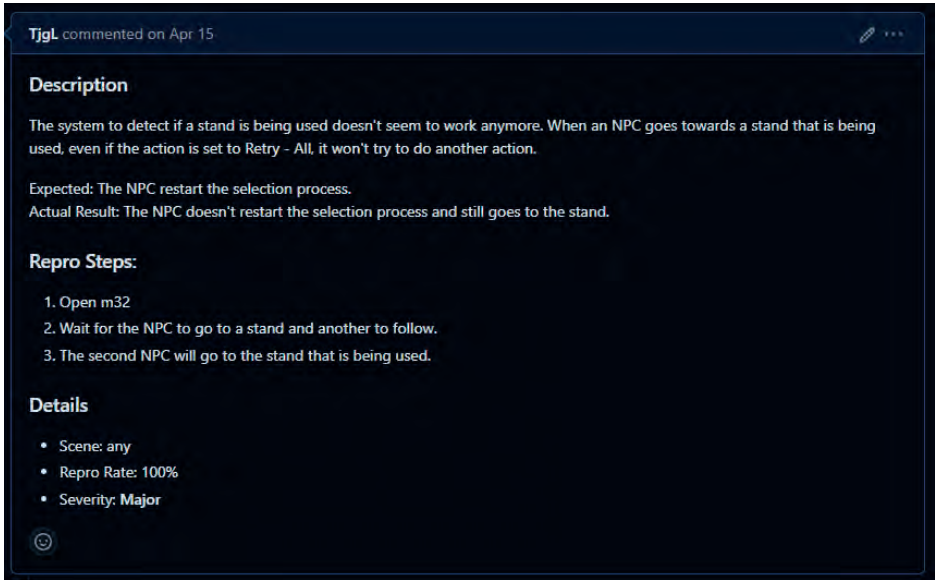
Bug Report

Suite à la semaine thématique de QA / UX, nous avons adapté notre fonctionnement vis à vis des bugs. Avant nous n'étions pas organisé, et dès que nous trouvions nous le disons au programmeur et il corrigeait rapidement le problème.

Mais à cause de cela, nous n'avions aucune tracabilité en dehors de «on a déjà eu ce bug là».

Mais depuis la fin de l'année, nous avons commencé à rapporter les bugs en tant qu'issue sur Github.

Ci-dessous, nous pouvons voir un exemple de description d'issue. Tandis que nous pouvons observer une liste d'issue corriger à droite.



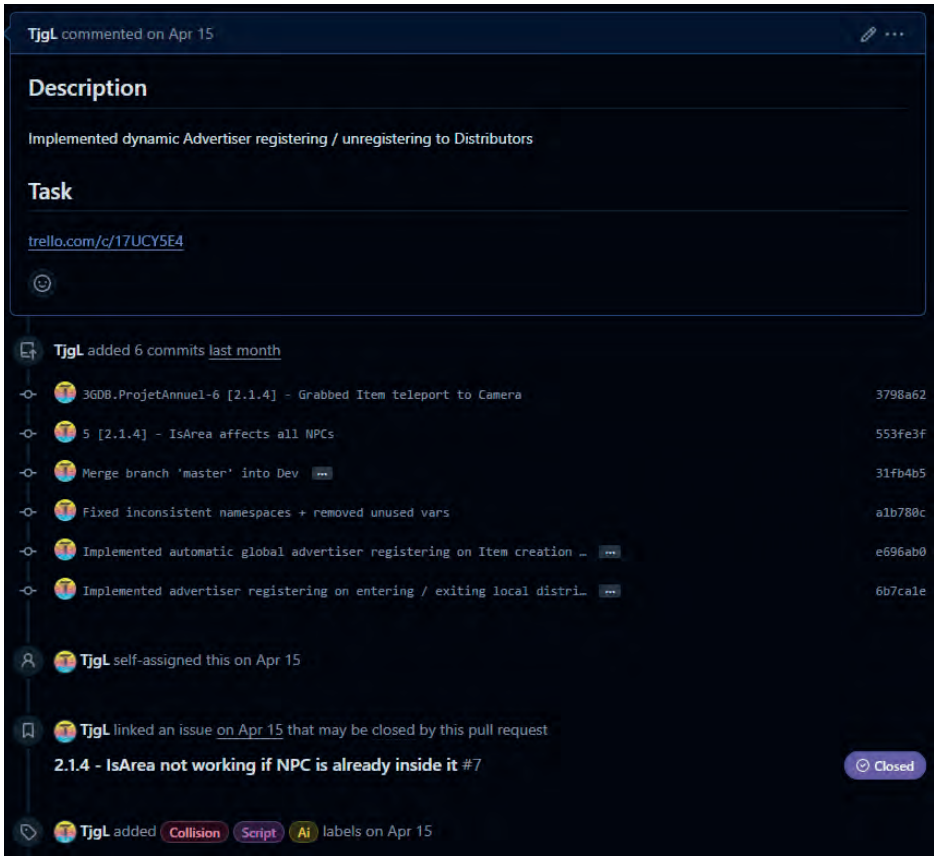
Dev Branch

Pour aller avec ce principe de bug report, nous avons commencé à travailler avec des branches de feature. C'est à dire que le développement de certaines fonctionnalité étaient fait sur des branches git pour garder une version stable sur le main. Cela a seulement été fait pour les features les plus importantes et complexes.

Pull Requests

Une fois que la feature était développé, une pull request était créée. Dès que le comportement de la feature était approuvée et que le prototype était testé, la branche était merge sur le main.

Ce qui est pratique avec les pull requests de Github c'est que l'on peut y lier des issues. Et dès que la pull request est merge, les issues liées sont automatiquement fermées. Ce qui est un gain de temps. Et c'est aussi satisfaisant de voir tout un groupe d'issue passer de open à closed.



Packages

Crédits

Notre projet à utilisé plusieurs packages lors de son développement et nous trouvons ça juste de lister ces derniers.

Addressables

Les Addressables est un package Unity permettant de charger des assets via une adresse.

Il nous a surtout aidé au niveau du système de chargement de scène.

Input System

L'input System est un package Unity remplaçant l'Input Manager.

Il nous a servit pour les inputs.

A Pathfinding Project Pro*

A* Pathfinding Project Pro est un asset développé par Aron Granberg. Il contient des components pour faire différents types de nav mesh et des agents.

Il nous a servit pour tout ce qui est navigation et local avoidance des npcs.

FMOD for Unity

FMOD for Unity est un package développé par FMOD permettant d'intégrer le moteur audio du même nom dans Unity.

Il nous a servit à rendre les sons plus intéressants dans le projet.

Realtime CSG

Realtime CSG est un package développé par Logical Error. C'est un outil de blocking basé sur les Constructives Solid Geometry.

Il a permit la réalisation rapide des différentes itérations des blockouts de notre prototype.

Steam Audio

Steam Audio est un asset développé par Valve. C'est un moteur audio permettant d'ajouter de l'occlusion, des reflections, de l'audio binaural et ambisonics dans différents moteurs de jeu.

A l'origine, nous utilisions ce package mais il ne fut plus nécessaire après le changement de notre prototype.

Amplify Shader Editor

ASE est un outil développé par Amplify Creation. C'est un éditeur de shader pour Unity.

Il nous a servit lors de la création de nos shaders.

Fullscreen Editor

Fullscreen Editor est un outil développé par Muka Schultze. Il permet à n'importe qu'elle fenêtre Unity d'être mise en plein écrans.

Hot Reload

Hot Reload est un outil développé par The Naughty Cult. C'est un package permettant de modifier en temps réel le code sans devoir le recompiler.

Il nous a permis de grandement réduire les temps entre les itérations.

Odin Inspector

Odin Inspector est un outil développé par Sirenix. Il ajoute différents attributs C# pour améliorer l'apparence de l'inspecteur.