Game Overview Document

AMICHI Hugo -ANTONIO Emma - ISLIWA Barthélemy - LEROY Matthias

2JVB - ICAN 2026

Sommaire

Présentation de l'équipe 3
Introduction 4
Fiche d'identité 5

Game Design
Intentions 8
3Cs 9
Système 12
Tendance système et tension 15
Schéma Système 16
Mécaniques (RGD) 17
Stratégies et utilisations mécaniques 19
Règles de jeu et objectifs 24
Boucles OCR 25
Références de Game Design 26
Diagramme de Ventrice 29
Métaboucle 30
Boucle de Gameplay 31
Situations de jeu 32
Boucles de prédiction 35

Tableau de Signes et Feedback 36
Tableau de Scope 38
Itérations 39

Direction Artistique 45
Intentions et références de départ 46
Thématique du Billard 47
Modernisation thématique 49
Stylisation 51
Couleurs 53
Boules 54
Feedbacks 56
Ambiances 59
Terrain de jeu 60
Background 61
UI 62
Logo 64
Shaders 65

Sound Design 67
Intentions et références 68
Réalisation des sons 69
Intégration dans FMOD 71

Programmation 73
Introduction 74
Enjeux techniques 75
I. La boule joueur 76
A.) PlayerScript - Mécanique principale 77
B.) PlayerScript - Impulsion et filtrage d'input 90
C.) PlayerScript - Gérer la vélocité 93
D.) Différenciation des joueurs et feedback du stick/impulsion 94
II. La boule 8 et le système de score 95
Evolution du rewind entre les versions 97

Fin de projet 101
Pistes d'amélioration 102
Conclusion 103
Remerciements 104

8 o'Clock - ICAN - 2025-2026

Présentation de l'équipe

Hugo AMICHI

Sound Design
Programmation
Intégration

Emma ANTONIO

Direction Artistique
Game Design
Documentation

Barthélemy ISLIWA

Programmation
Game Design
Documentation

Matthias LEROY

Game Design
Level Design
Game Art 3D

Introduction

Dans le cadre de notre projet de deuxième semestre en 2e année en Game Design au sein de l'ICAN, nous avons eu l'opportunité de concevoir un jeu à partir d'un jouet, grâce à la méthodologie Bottom-Up en pensant le jeu comme système.

Le thème pour ce projet était le Temps, un phénomène à la fois omniprésent et immatériel qui a guidé nos intentions durant l'entièreté du projet.

Lors de ce semestre, il a été également l'occasion pour nous d'améliorer nos compétences déjà acquises au premier semestre.

À travers cette documentation, nous espérons pouvoir vous retransmettre au mieux nos différentes réflexions, itérations et cheminements qui ont amélioré nos capacités de design au cours de ce second semestre.

Bonne lecture à vous !

Fiche d'identité

*La tension est à son comble : deux joueurs, une table de billard, et la dernière boule n°8 à remporter. Dans **8 o'Clock**, manipulez le cours du temps pour déplacer votre boule et changez le cours de la partie afin de remporter la victoire dans ce jeu mélangeant Billard et Air Hockey !*

Genre

Versus/Arcade

Plateforme

PC (Manette)

Cible

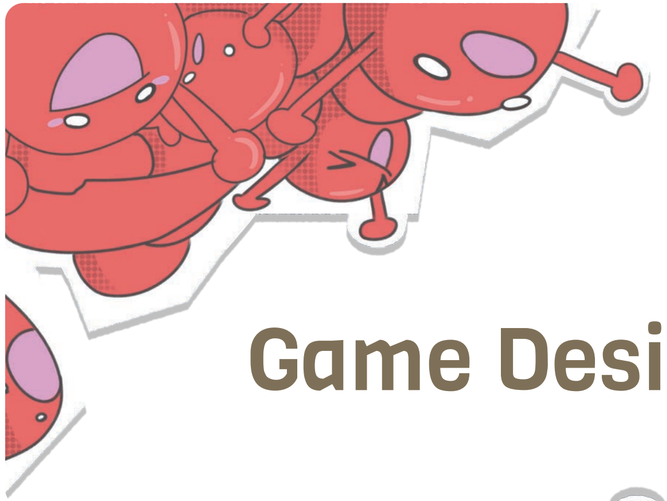
Casual à midcore, adeptes de petits jeux de versus, fans de sport et amateurs de mécanique de contrôle originale.

Univers

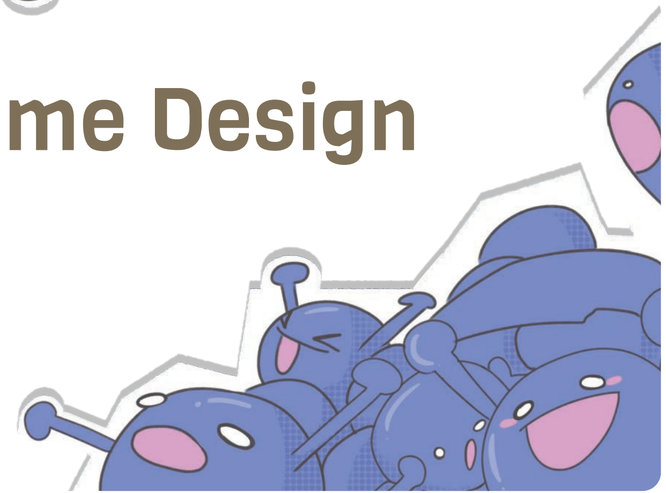
Mélange de Billard et de Air Hockey, le tout dans une esthétique pop en 2D/3D

Game Concept

Dans **8 o'Clock**, deux joueurs s'affrontent : leur objectif est de venir remporter la dernière boule d'une table de billard, la 8-ball. Ils peuvent pour cela manipuler la trajectoire passée de leur boule, leur permettant de venir se placer à des endroits stratégiques afin de prendre le contrôle de la partie. Le jeu se joue en 3 manches, avec 5 points à marquer pour en remporter une : le premier joueur à atteindre les 3 manches remporte la partie !



Game Design



Intentions

Notre intention principale pour ce projet était l'obtention d'un cœur mécanique émergent, qui permettrait d'obtenir un panel varié de déclinaisons possibles, dans l'optique initiale d'un Puzzle Game.

Cependant, au fil de nos itérations, nous avons décidé de recentrer le noyau de notre jouet autour de son émergence axée synchronisation et création de nouvelles situations de jeu, en prenant référence sur des jeux comme le Golf ou le Billard. Après avoir itéré de nombreuses fois, nous sommes arrivés à un jeu de type arcade permettant au mieux de rendre compte du plaisir de manipulation de la mécanique, sans exiger du joueur une maîtrise parfaite.

L'idée était de pouvoir, à travers notre projet, provoquer un effet de surprise auprès du joueur, en réalisant l'un des principaux fantasme de l'homme : pouvoir contrôler le temps.

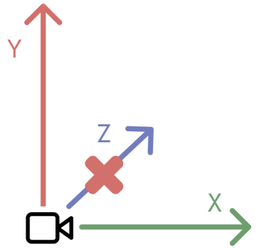
3Cs

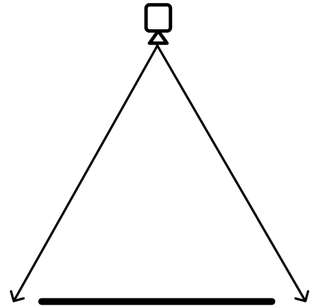
Caméra

La caméra de **8 o'Clock** est une caméra fixe en vue de haut (Top-Down), qui permet de rendre compte de l'entièreté de la zone de jeu.

Dans le cadre d'un jeu de versus où deux joueurs s'affrontent pour prendre le contrôle d'un objet commun, il était important que chaque joueur aie une vue complète sur l'état du système : en ayant une caméra fixe qui montre l'intégralité de l'écran de jeu, les deux joueurs ont accès à tous les éléments d'analyse nécessaires (trajectoire personnelle ou ennemie, position de la 8-Ball) pour établir leur plan de jeu.

Cette décision a également été appuyée par des références comme **Windjammers** ou **Pong**, où les deux joueurs ont une vue intégrale sur la zone de jeu.





Metrics

- **Résolution de l'écran** : 16:9
- **Projection orthographique** : 13
- **Distance avec le terrain** : 29m

Character

Dans **8 o'Clock**, chaque joueur contrôle un boule de billard. Les boules des deux joueurs sont différenciables à travers leur couleur, leur motif (pleine ou rayée) ainsi qu'à travers le visage présent dessus.

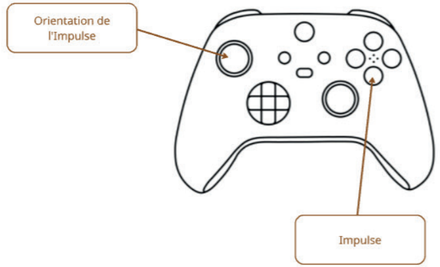
Le joueur ne peut pas décider directement de la position de sa boule, mais peut la déplacer sur la table en utilisant :

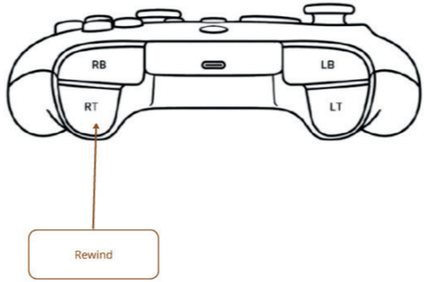
- le Rewind, qui permet de rejouer la trajectoire précédente de la boule sur le terrain ;
- l'Impulse, qui permet au joueur de projeter la boule dans une direction de façon périodique.

Grâce à ces deux mécaniques, le joueur peut déplacer sa boule sur le terrain dans l'optique de provoquer des collisions ou de stopper des trajectoires.

Enfin, chaque joueur possède un score visible dans l'UI en haut de l'écran, et qui s'incrémente lorsque le joueur a réussi à marquer la 8-Ball dans le camp adverse.

Contrôles





8 o'Clock est un jeu se jouant à la manette :

- la Gachette arrière droite (RT/ZR/R2) permet de contrôler le Rewind en fonction de la pression appliquée (+ le joueur appuie fort, + son rewind va être rapide) ;
- le Joystick Gauche + le bouton du bas (B/A/X) permettent de contrôler l'Impulse : le joueur indique la direction dans laquelle il souhaite projeter sa balle avec le Joystick, et confirme l'input avec le bouton du bas. En fonction du maintien du bouton, la puissance de projection est amplifiée.

La décision de concevoir le jeu pour une utilisation manette a été motivée par des enjeux de Gamefeel et d'utilisation du controller : la manette renvoie des valeurs analogues, permettant ainsi d'avoir un effet de mesure pour le Rewind.

De la même façon, nous avons également priorisé un controller avec peu d'inputs mais une grande marge de manipulation à un controller avec beaucoup de boutons situationnels : avoir peu d'inputs à retenir allège la charge mentale du joueur, et lui permet de se concentrer sur la manipulation de la mécanique principale.

Système

Boules Joueur

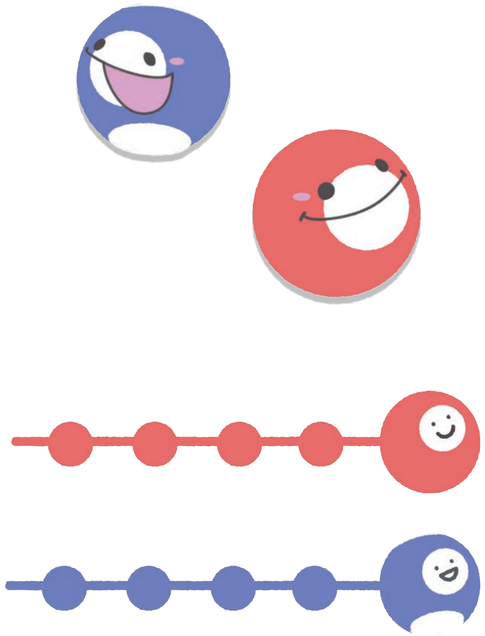
Les boules joueur sont les principaux éléments du système de **8 o'Clock** : chacune contrôlées par un joueur, ce sont elles qui vont amener le plus de modifications dans le système.

Caractéristiques particulières

- Déplacement sur les axes XZ
- Force de friction : 0.6 (Dynamic) + Bouncyness de 0.8
- Vitesse contrainte à une valeur max (20), si <1 = Nul
- Poids : 1 Kg

Les metrics des boules joueur ont été pensées pour avoir un comportement similaire aux boules de billard, que ce soit à travers leur vitesse ou leur changement de trajectoire lors de collisions avec d'autres éléments (notamment les bordures du terrain).

Se faisant, il est possible pour le joueur de contrôler leur déplacement en faisant appel à des principes mathématiques : en provoquant une collision à un angle précis, il est possible de prévoir la trajectoire de la balle. Un tel niveau de maîtrise n'est pas attendu des joueurs, mais ouvre la porte à des possibilités de Mastering.



8-Ball

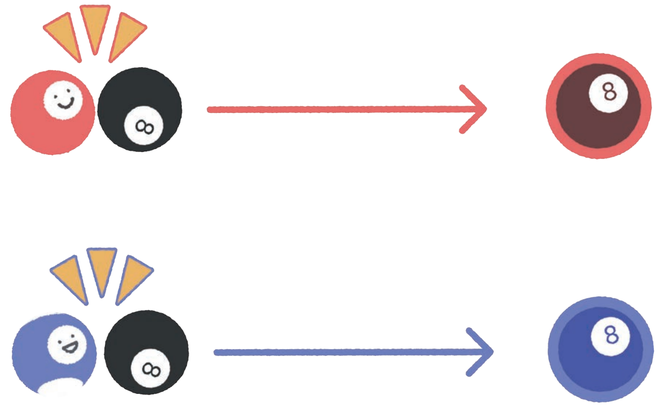
La 8-Ball est un élément central du système : c'est autour de ses collisions et de sa position dans l'espace que s'articulent les objectifs de jeu.

Caractéristiques particulières

- Déplacement sur les axes XZ
- Force de friction : 0.6 (Dynamic) + Bouncyness (0.6)
- Poids : 0.1 Kg

Comme pour les boules joueurs, ses metrics ont également été pensés pour avoir un comportement proche des boules de billard, avec comme nuance principale son poids moins élevé que les boules joueur.

En fonction du dernier joueur ayant rentré en contact avec, sa couleur change : rougeâtre ou bleutée. Enfin, lorsqu'elle rentre en contact avec le fond d'un goal, elle incrémente le score du joueur adverse.



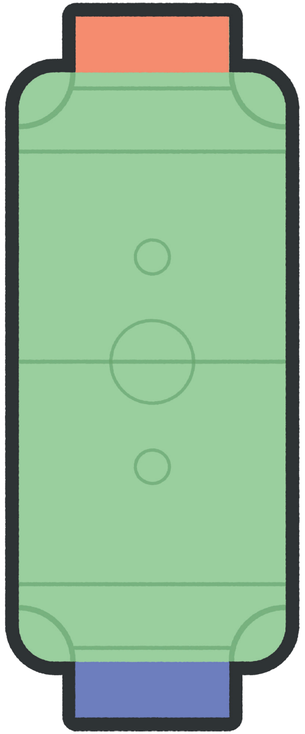
Terrain et Goals

Le terrain du système (aussi appelé Table) est composé de 2 éléments principaux :

- le centre, dans lequel la majorité du jeu va se dérouler. La 8-Ball apparaît au centre de cette zone ;
- les Goals, situés aux deux extrémités du terrain avec un Goal par joueur. En début de partie, les deux sont inaccessibles : en fonction du joueur ayant touché en dernier la 8-Ball, le Goal adverse s'ouvre pour permettre au joueur de marquer. Ce fonctionnement permet d'éviter les cas d'autobut et de concentrer davantage le jeu autour de la 8-Ball.

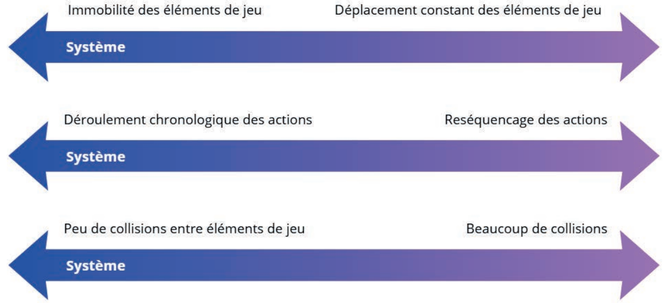
Ses bordures permettent au joueur de réaliser des collisions particulières dans le même esprit que le billard, où l'on utilise les bords de la table pour provoquer des trajectoires spécifiques.

Sa forme est pour l'instant très simple : un rectangle de 40m x 20m, avec les deux goals de 10m aux extrémités : nous avons envisagé la possibilité de proposer différentes formes de terrain (avec notamment des variantes où des parois présentes sur la table amèneraient des situations de jeu particulières), mais pour des questions de scope, nous avons préféré nous concentrer sur un terrain unique pouvant être mieux itéré.



Tendance système et tension

Le système de 8'o-Clock tend vers une situation de jeu immuable se traduisant par l'immobilité des éléments mouvants le composant, càd les boules. De la même façon, sans la mécanique de Rewind, le système tend vers un déroulement chronologique des actions : les probabilités qu'une boule joueur réatteigne un état dans le système précédent sont quasiment nulle. Enfin, les éléments tendent à ne pas être synchronisés, et les collisions à être provoquées par la coïncidence de deux objets entre eux.

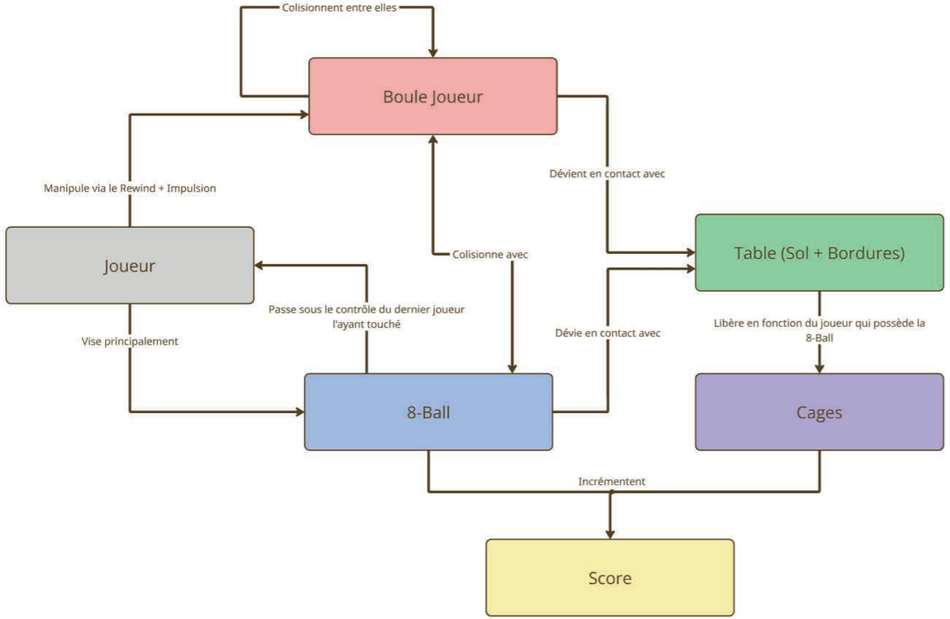


Les joueurs s'opposent à cette tendance à travers l'utilisation du Rewind et de l'Impulse :

- l'Impulse permet de mettre en mouvement la boule du joueur, amenant à des collisions dans la zone de jeu ;
- le Rewind permet quant à lui de replacer la boule dans un état précédent (position et vitesse incluse) : se faisant, le reste du système n'ayant pas été sujet à ce retour dans le temps, lorsque le joueur relâche le Rewind, une nouvelle situation de jeu se produit.

La tension système réside notamment dans l'opposition entre la capacité du joueur à ramener sa boule dans un état précis à des fins de contrôle, avec le reste des éléments du système qui eux évoluent de façon linéaire : plus le joueur va chercher à synchroniser sa boule avec les autres éléments du système (8-Ball et boule du joueur adverse), plus il va venir s'opposer au système qui est de base asynchrone.

Schéma Système



Mécaniques (RGD)

Rewind

Mécanique

La boule retrace le chemin parcouru pendant les 8 dernières secondes avant le début du Rewind, supprimant ainsi les sauvegardes qu'il traverse. À la fin du Rewind, les forces (direction, vitesse, rotations...) de la sauvegarde la plus récente lui sont appliquées.

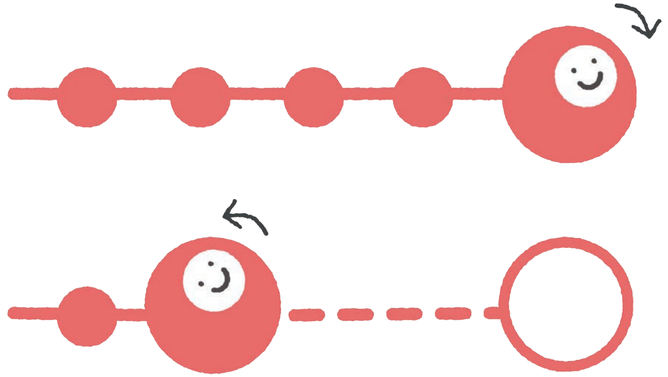
Input : RT (Xbox) / ZR (Nintendo) / R2 (Playstation)

Paramètres

- Cooldown : 0s
- Vitesse de rembobinage : 50 sauvegardes/s (Echelle 1) x Valeur analogue de la gachette (de 0 à 1) x Valeur de filtrage

Feedbacks

- Son mécanique
- Réduction/Agrandissement et emplacement de la trail
- Déplacement de l'objet



Impulse

Mécanique

La boule est propulsée dans la direction indiquée par le joystick à une force variable en fonction du temps de maintien du bouton.

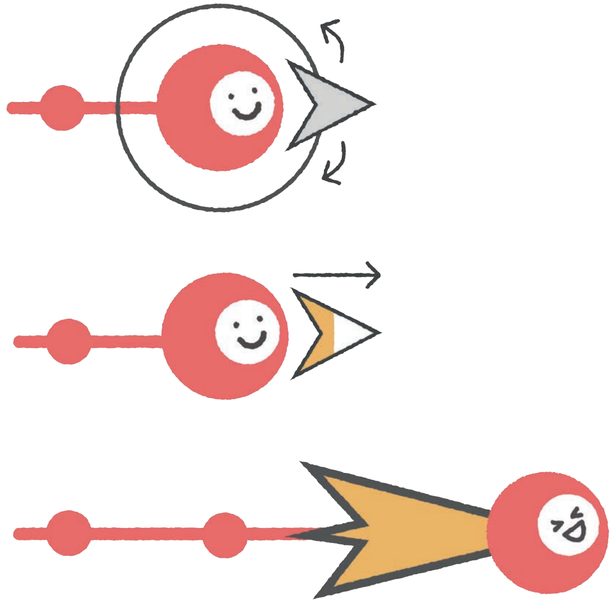
Input : Joystick Gauche + A (Xbox) / B (Nintendo) / X (Playstation)

Paramètres

- Cooldown : 3s
- Puissance de propulsion : 2 x 2 x Temps de maintien du bouton (0 à 1/deltaTime)

Feedbacks

- Curseur dont la position dépend de l'axe de rotation du stick du joueur
- Trail spécial derrière l'objet
- Déplacement de l'objet
- Durée du Cooldown + Force de charge indiquée par une jauge dans le curseur



Pourquoi cette mécanique ?

Cette mécanique permet au joueur, à des moments clés, d'obtenir plus de contrôle sur sa boule, mais aussi de s'enregistrer de nouvelles trajectoires à exploiter ensuite avec le Rewind.

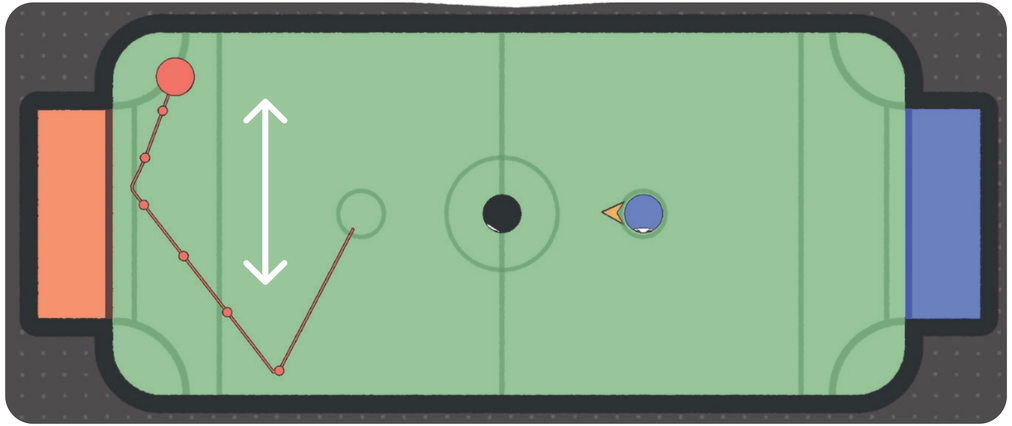
Stratégies et utilisations des mécaniques

À travers des playtests auprès de différents types de joueurs, nous avons pu établir une liste des principales stratégies et méthodes employées par les joueurs lors de leur utilisation des deux mécaniques.

Rewind

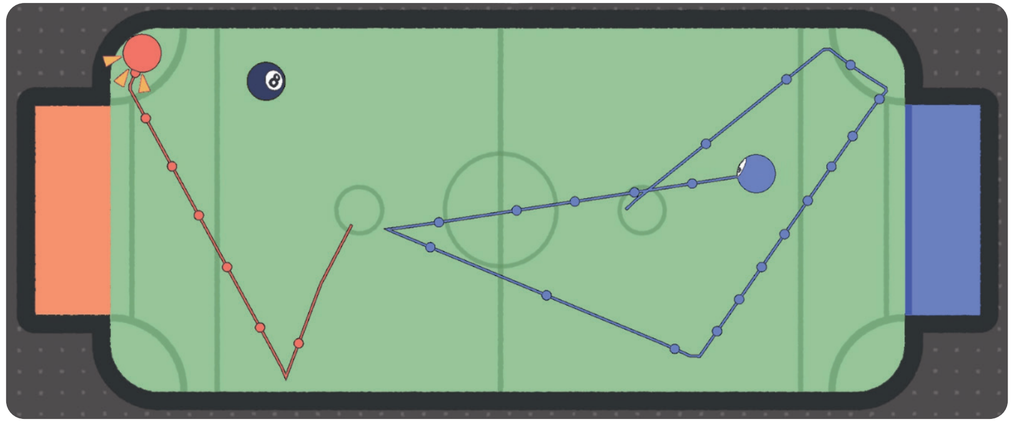
Zonning

Le Rewind peut être utilisé par certains joueurs pour couvrir une portion particulière du terrain. En alternant rapidement appui et relâchement de la gâchette, le joueur peut faire un aller-retour sur une portion de sa trajectoire. Cela lui permet par exemple de venir protéger une zone, notamment son goal, à la manière du babyfoot.



Stabilisation

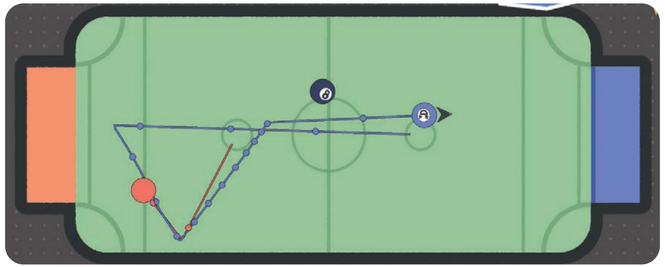
De la même manière, en alternant les appui encore plus rapidement, le joueur peut stabiliser son objet à un instant du Rewind : il peut s'en servir pour ensuite projeter sa boule depuis un endroit précis, pour pour faire obstacle au joueur adverse à un endroit clé de son Rewind.

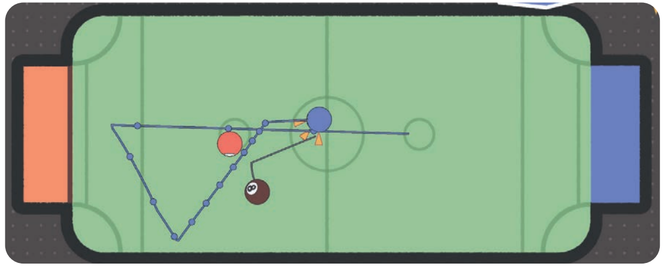


Inversion

Enfin, la balle pouvant également provoquer des collisions pendant le Rewind, certains joueurs utilisent la mécanique afin d'inverser une force : après s'être projeté dans un sens, ils utilisent le Rewind pour pousser un objet depuis l'autre côté.

Cela s'avère utile dans certaines situations précises, notamment lorsque l'on est opposé au Goal que l'on doit viser.

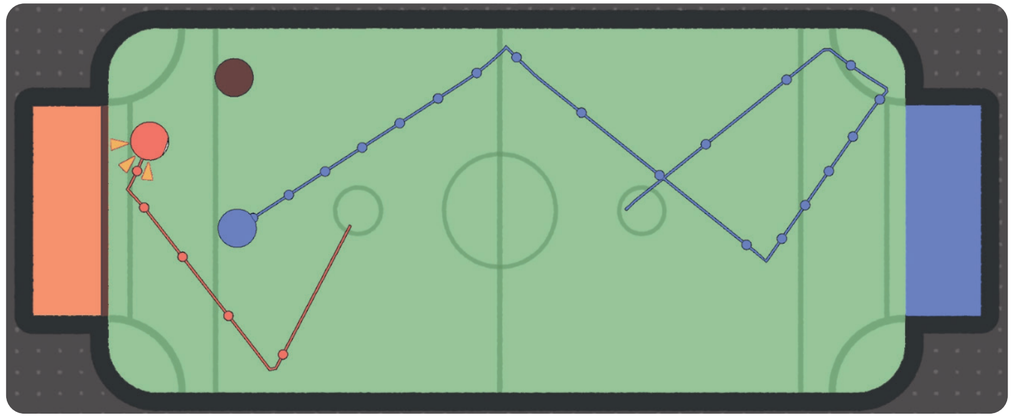


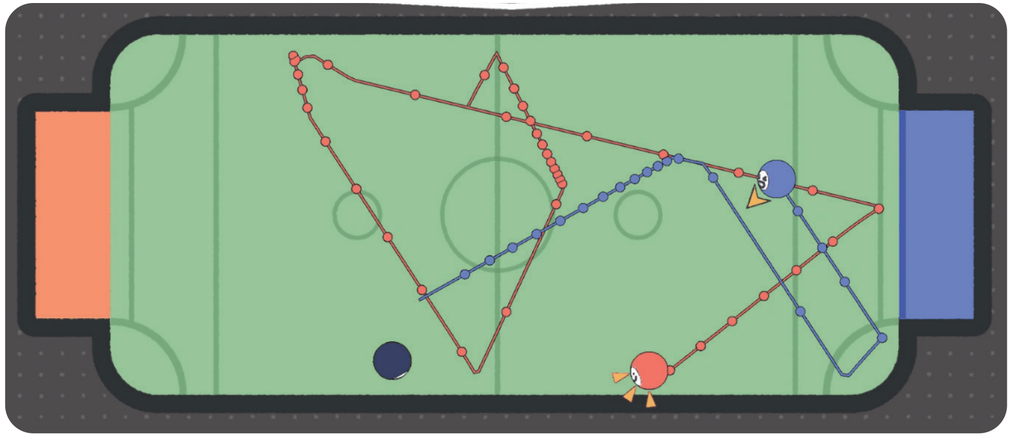


Impulse

Impulse en V

Certains joueurs avancés utilisent l'Impulse pour provoquer une collision à un angle particulier avec une bordure du terrain, qui permet de faire parcourir à leur boule une grande surface en forme de V. Cette utilisation de la mécanique se rapproche d'éléments présents dans le Billard.





Impulse vers le joueur adverse

D'autres joueurs utilisent également la mécanique afin de directement se projeter sur le joueur adverse, notamment pour perturber leurs trajectoires déjà enregistrées.

Cette utilisation de la mécanique s'apparente à du sabotage, et nous avions considéré pendant un temps de la contraindre : néanmoins, c'est une partie inhérente du jeu et une autre façon d'utiliser la mécanique à son avantage, et nous avons fait le choix de ne pas la pénaliser, étant donné qu'il s'agissait juste d'une utilisation différente de la mécanique.

Petit ou grand impulse

Enfin, il y a également une séparation des joueurs en deux groupes : ceux qui préfèrent charger une grande Impulse pour couvrir plus de terrain, et ceux qui favorisent de petites impulsions pour les sortir plus rapidement.

Règles de jeu et objectifs

Règles du jeu

Après avoir identifié les différentes utilisations du toy, nous avons pu définir les règles du jeu : bien que servant de prétexte pour amener à la manipulation de la mécanique, nous avons fait en sorte que ces règles soient adaptées à la cible en fonction du temps de jeu demandé.

Nous en sommes arrivés à un système de manches et de points :

- Un joueur gagne lorsqu'il remporte 3 manches ;
- Une manche est remportée lorsqu'un joueur marque 5 points.

Entre chaque manche, les sauvergardes des deux joueurs sont supprimées et ils sont retéléportés au centre de la table : cela permet, entre chaque manche, qu'il y ait une remise à 0 de l'état de jeu des deux joueurs pour permettre plus facilement une remontée du joueur perdant et éviter les trop gros écarts.

Objectifs

En fonction de la situation de jeu et des différentes échelles, nous avons défini les différents objectifs présents dans le jeu :

Micro

- Atteindre une position spatiale particulière ;
- Provoquer une collision avec une bordure, le joueur adverse ou la 8-Ball.

Mezzo

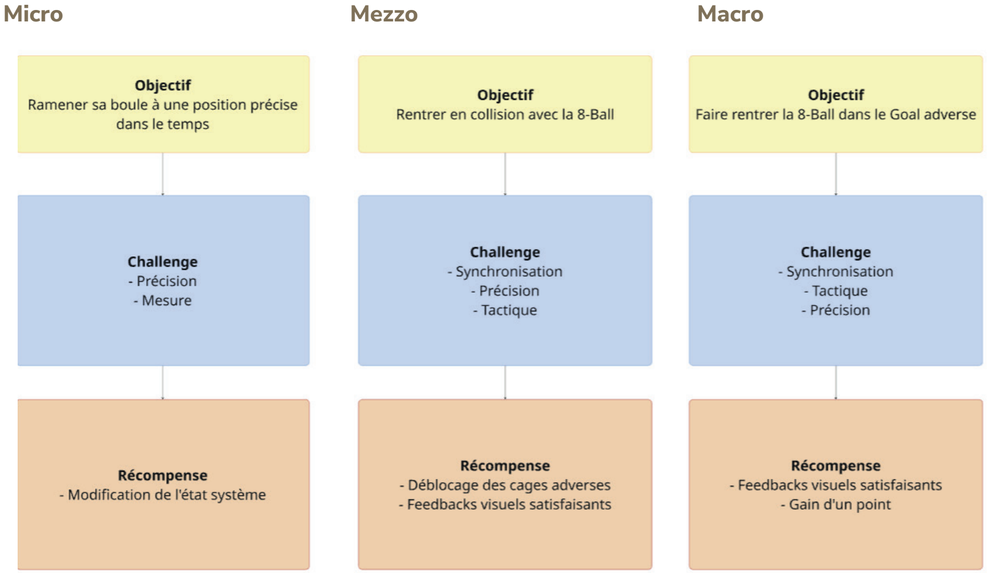
- Obtenir une trajectoire particulière ;
- Bloquer une trajectoire adverse.

Macro

- Amener la 8-Ball dans le Goal adverse ;
- Remporter une manche (= 5 points) ;
- Remporter la partie (= 3 manches).

Ces différents objectifs sont imbriqués les uns dans les autres et sont pour la plupart auto-défini par le joueur en fonction de son plan de jeu.

Boucles OCR



Références de Game Design

The Entropy Centre

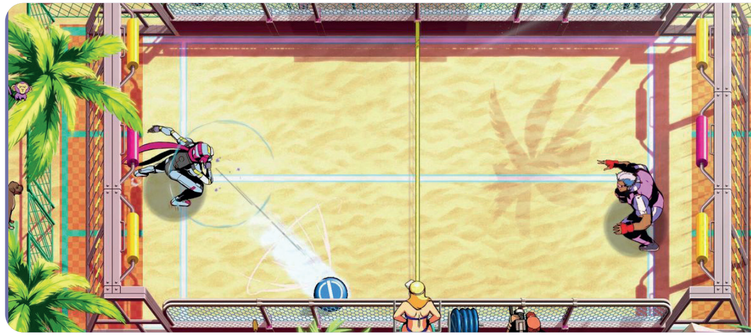
Ce jeu a été l'une de nos références initiales quant à l'exploitation de notre mécanique : lors du passage de la V1 à la V2, alors que nous étions en train de questionner quel angle prendre pour développer notre mécanique, nous avons pu nous référer à ce jeu pour voir comment une mécanique similaire était utilisée dans le cadre d'un puzzle game.

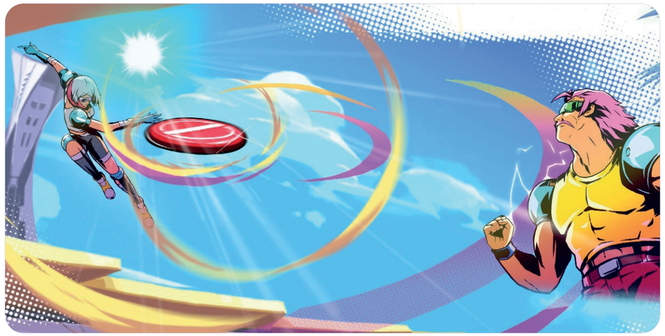
La principale différence entre ce jeu et notre projet est l'enregistrement des forces appliquées à l'objet, qui nous permet de jouer sur la force donnée ou reçue d'un objet, là où The Entropy Centre ne permet de rembobiner qu'une position dans l'espace.

Cette différence en tête, nous avons réalisé qu'il y avait un potentiel lié à la réalisation de trajectoires à partir d'objets synchronisés, et nous avons ainsi pu explorer cette idée en détail.

Windjammers

Windjammers a été une référence directe par son design de jeu Versus : il s'agit d'un jeu nerveux, rempli de feedbacks visuels et audios. Chaque action devient démesurée et leur impact est démultiplié : nous nous sommes servis de cette référence pour donner à notre jeu cet aspect juicy dans les feedbacks, afin que le joueur se sente puissant derrière la manette.





Nous nous sommes également servis de cette référence pour le placement de la caméra : la caméra vue Top-Down permet une meilleure lisibilité de l'information, ce qui était nécessaire pour notre jeu par sa tendance à saturer l'écran de jeu ; en Top-Down, l'information reste claire et constamment disponible pour le joueur.

Nous nous sommes aussi inspirés du système de score pour son positionnement, mais aussi sa hiérarchie de l'information dans l'UI, compréhensible en un regard sans besoin d'explication pour le joueur.

Subpar Pool

Ce jeu est une référence centrale en termes de Game Design par sa proximité avec le billard, en étant un exemple de jeu employant cette thématique dans ses mécaniques.

Il était important pour nous de donner au joueur un sentiment de réalisme pour ce qui est des collisions ou encore le comportement des boules : nous nous sommes inspirés de la physique du jeu pour que le jeu soit plaisant en termes de game feel, mais aussi visuellement.

L'une des majeures différences entre ce jeu et notre projet vient des feedbacks visuels de trajectoire : dans ce jeu, la trajectoire illustrée est celle à venir de la boule, tandis que dans notre projet, cela correspond à la trajectoire effectuée.

Cependant, malgré ce changement, nous sommes restés sur un visuel similaire pour faire comprendre facilement au joueur qu'elle correspond à un mouvement.

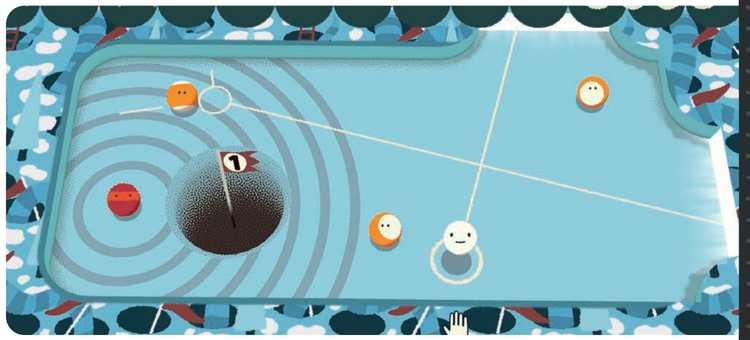
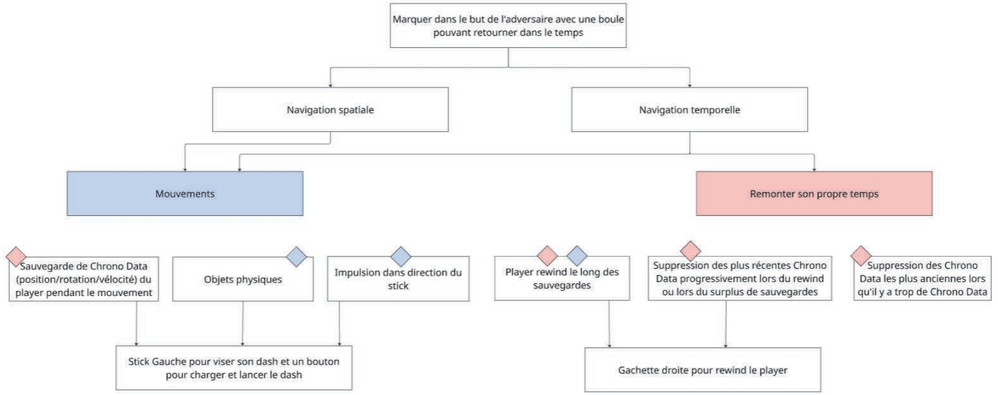
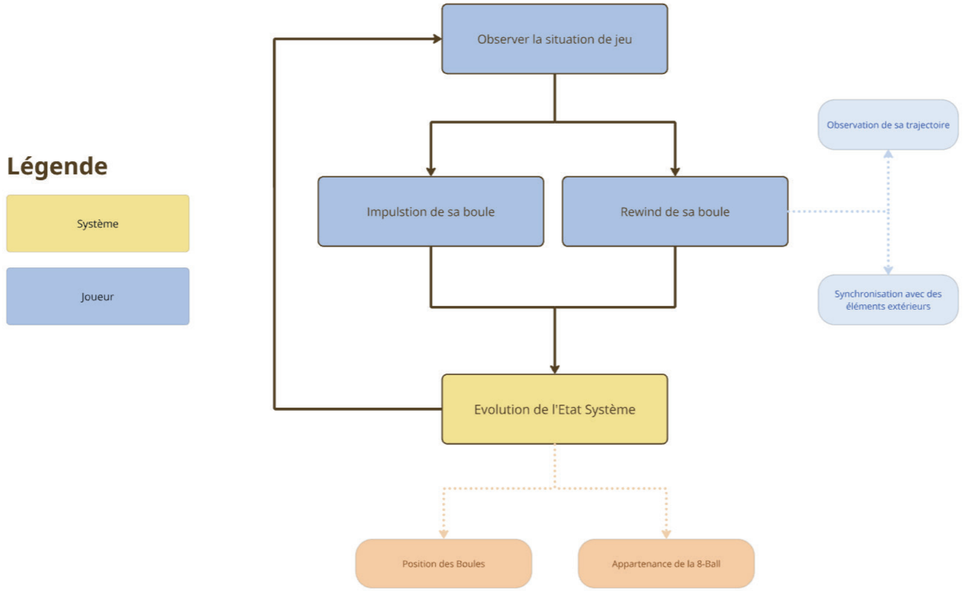


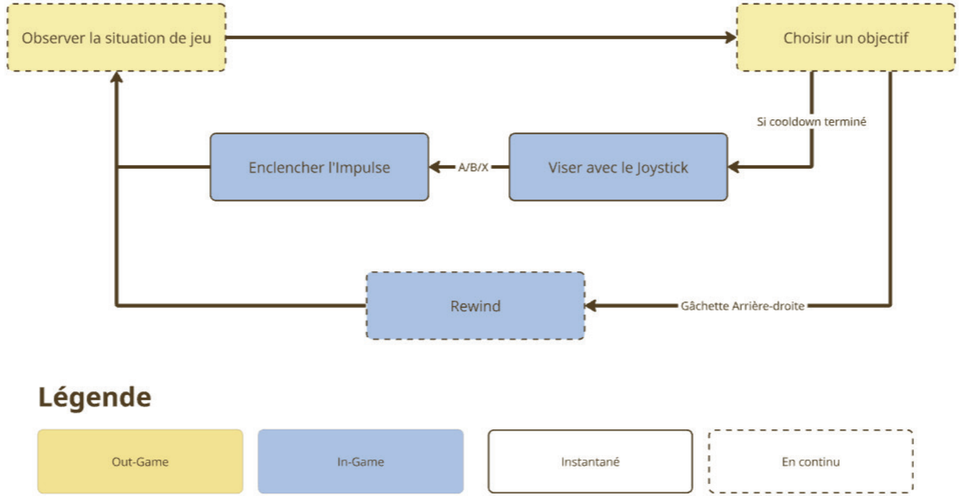
Diagramme de Ventrice



Métaboucle

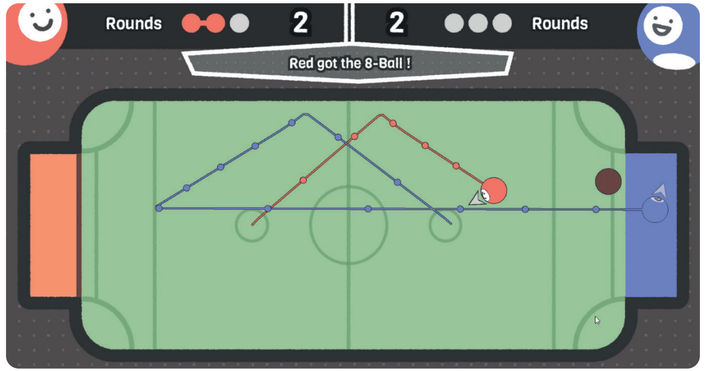


Boucle de Gameplay



Situations de jeu

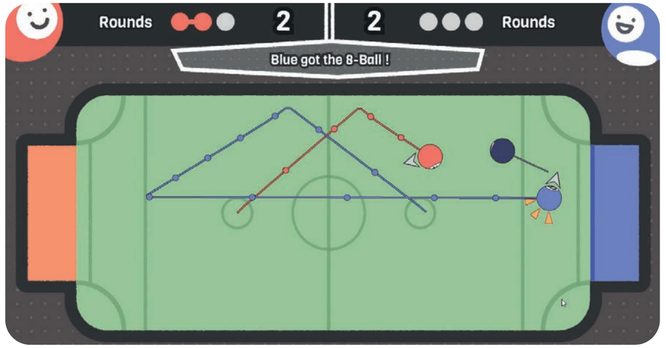
Dans cette situation de jeu, le joueur rouge est en avance sur le joueur bleu avec 2 rounds à son compte, et les joueurs sont à 2 points chacun. La couleur de la 8-ball et le message au dessus indique une collision récente du joueur rouge ; la position de la boule bleue indique également que le joueur est en position de défense de son propre but.



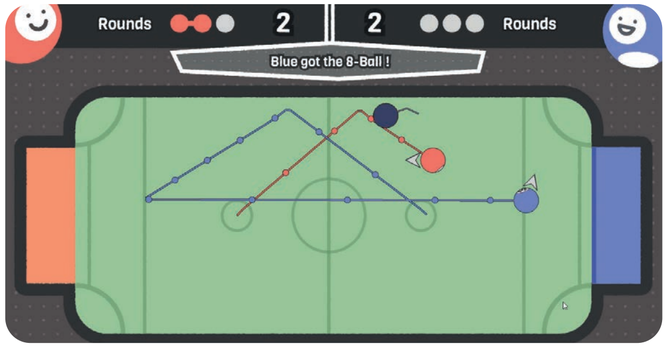
Cependant, la position de la boule bleue peut laisser une ouverture au tir du joueur rouge : les trails de rewind indiquent les déplacement disponibles pour les 2 joueurs. Le joueur rouge, lui, pourra facilement retourner dans son camp au vu d'une contre attaque ; quant au joueur bleu, il pourra se rendre rapidement dans le camp adverse.

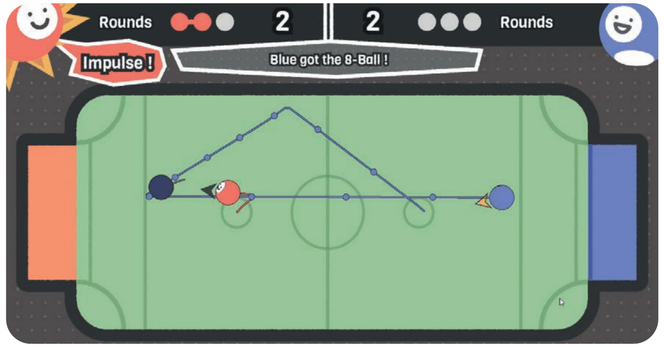
L'impulse est disponible pour les 2 joueurs permettant de s'avancer en direction de la balle ou de gêner l'ennemi.
Nous sommes ici dans une situation particulièrement favorable pour le joueur rouge.

Pour défendre son but, le joueur bleu a décidé de rewind pour collisionner avec la 8-ball ; nous pouvons observer que la trail de la 8-ball est à son maximum, indiquant un tir puissant de la boule bleue. La direction de la 8-ball prédit une future collision sur le mur en forme de « V « . Le joueur rouge dispose encore de son Impulse. Le but du joueur rouge s'ouvre et la situation devient alors plus favorable pour le joueur bleu.

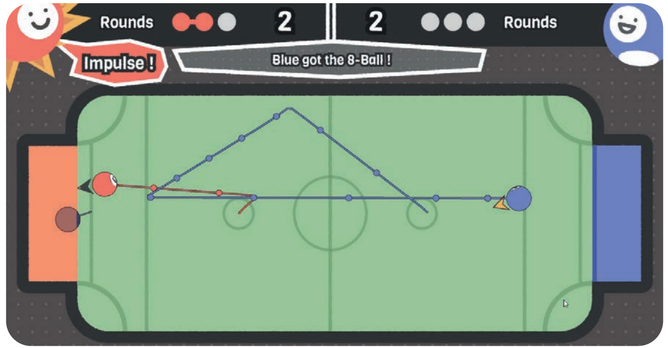


Comme nous pouvons observer, la 8-ball dépasse le joueur rouge : l'angle de la trail de la 8-ball indique qu'elle se dirige droit dans le but rouge. Ici, le joueur bleu est encore en rewind. Le revirement de situation devient alors intéressant pour le joueur bleu. Malgré le tir précis du joueur bleu, la trail de rewind du joueur rouge passe parfaitement sur la trajectoire de la 8-ball. Le joueur rouge n'est donc pas dans une situation de but inévitable.





Ici, nous pouvons observer que la vitesse de la boule bleue a surpris le joueur rouge : malgré une tentative de rewind pour toucher la 8-ball, cela n'a pas suffit pour la faire changer de couleur. Nous pouvons observer que le joueur a utilisé son mpulse en direction de la 8-ball grâce aux feedbacks visuels pour tenter de sauver son but. Le joueur bleu, lui, préfère rester dans son camp pour parer à une éventuelle contre-attaque.



Sur cette frame, nous pouvons observer que le joueur rouge a manqué la 8-ball malgré une tentative de dernière seconde : la couleur et la position de la 8-ball indique très clairement un but en faveur pour le joueur bleu, lui même resté en retrait pour garder une bonne position pour le début du prochain point.

Boucles de prédiction

Prédiction	Décision	Action	Régulattion	Apprentissage
Je veux tirer dans le but adverse	Je choisis d'impulse en direction de la balle	Je dirige le Left Stick dans la direction souhaitée et je maintiens/relâche le bouton	L'adversaire Impulse avant moi et collisionne avec la 8-ball, je loupe mon tir	J'apprends que je dois prendre en considération la position et la disponibilité avant d'engager un tir
Je veux positionner une trail devant mon but pour défendre une future attaque	J'impulse dans la hauteur du terrain devant le but	Je dirige le Left Stick dans la direction souhaitée et je maintiens/relâche le bouton	Je touche la balle, sa couleur change mais malgré cela, elle continue vers le but.	J'apprends que lorsque la 8-ball est teinté de ma couleur, mes buts se ferment rendant la défense un peu plus facile
Je veux dégager la 8-ball dans la moitié de terrain adverse	Je choisis d'impulse en direction de la balle	Je dirige le Left Stick dans la direction souhaitée et je maintiens/relâche le bouton	L'adversaire contre la 8-ball et reviens directement en direction de mes buts sans passer par ma trail	J'apprends que l'Impulse possède un timer et qu'il faut l'utiliser en fonction de son positionnement initial mais aussi celui de l'adversaire
Je veux me repositionner sur le terrain	Je choisis de rewind	Je maintiens la gâchette pour me positionner à l'endroit voulu	Malheureusement, je rewind trop et ma balle par dans une direction non souhaitée	J'apprends que la balle sauvegarde le mouvement et la vélocité à un moment donné. Si je rewind avant une collision avec le player, le perds la sauvegarde de cette collision si je rewind trop.
Je veux empêcher le joueur adverse de marquer	Je choisis de rewind	Je maintiens la gâchette pour me positionner à l'endroit voulu	Lors du rewind, le joueur avec collisionne avec moi et se voit propulsé	J'apprends que les collisions entre joueurs fonctionnent aussi pendant le rewind

Tableau de Signes et Feedback

Mécanique	Parties intéragissantes	Signe	Action	Feedback	Priorité

Collision Joueurs	Joueurs	Bonne trajectoire de tir pour l'adversaire	Le joueur cherche la collision avec l'adversaire	Feedbacks visuels :
				• Apparition de 6 triangles jaunes
				  • 3 autour du player 1
				  • 3 autour du player 2
				Feedbacks audio :
				• Petit son mixé d'une collision de boules de billard
					Moyenne

But	8ball et espace de jeu	La 8ball imbibée d'une couleur se dirige droit dans le but adverse	/	Feedbacks visuels : • Screenshake • Apparition d'UI supporter pour le joueur ayant marqué • Phrase centre/haut de l'écran annonçant le player ayant marqué • Points côté joueur augmente de 1 jusqu'à 5 Feedbacks Audio : • Bruit de validation mécanique • Foule en délire	Haute

Collision joueur Rewind/ joueur non rewind	Joueurs	Trails se superposent	Le joueur retourne dans le temps pour revenir à une position voulue	Feedbacks visuels :
				• Apparition de 6 triangles jaunes
				  • 3 autour du player 1
				  • 3 autour du player 2
				Feedbacks audio :
				• Petit son mixé d'une collision de boules de billard
					Haute

Rounds	UI	Manche gagnée par un joueur	/	Feedbacks visuels : • 1 des trois boules grises du du camp remportant la manche se remplie de la couleur du player	Haute

Mécanique	Parties intéragissantes	Signe	Action	Feedback	Priorité
Déplacement	Joueur et espace	/	/	Feedbacks visuels : • Rotation de la boule sur le sol	Haute
Coup de boost	Joueur et espace	Icone du boost débloquée	Input A Viser avec Joystick Gauche	Feedbacks visuels : • Bulle de Texte apparaît coté camp joueur écrit : Impulse ! • Onde choc visuel derrière UI du joueur Feedback Audio : • Petit son de bois avec bruit exagéré de vent	Haute
Rewind	Joueur	Trail et hologrammes	R2	Feedbacks visuels : • Trail et hologramme • Rotation de la boule dans le sens inverse Feedback audio : • Son digital rembobinage	Haute

Collision Joueur et 8ball	Joueur et 8ball	Bonne trajectoire de tir disponible	Le joueur touche la 8ball	Feedbacks visuels :
				• 8ball se teinte de la couleur du joueur collisionné
				• Apparition de 6 triangles jaunes
				  • 3 autour de la 8ball
				  • 3 autour du player
				• Phrase centre/haut de l'écran annonçant à qui appartient la 8ball
				Feedback audio :
				• Petit son mixé d'une collision de boules de billard
					Moyenne

Tableau de Scope

	Noyau	Incrémentation A	Incrémentation B
Contrôles	Clavier à partager pour 2 personnes	Prise en charge de 2 manettes en local	Prise en charge de 2 manettes en ligne
Joueurs	1vs1 Local	2vs2 Local	1vs1 / 2vs2 multijoueur
8-Ball	Réaction simple de la boule aux collisions	Amélioration de la Physique et comportement plus réaliste	Plusieurs variation de 8-Ball associées à des difficultés différentes
Rewind	Rewind à vitesse fixe	Rewind adaptatif d'après la pression de la gâchette	Rewind modulaire modifiable par les joueurs dans les paramètres
Level Design	Un seul terrain fixe	Plusieurs terrains de formes variées plus complexes	Ajout d'objet spéciaux disséminés sur la table amenant des changements dans les collisions
Stylisation	Modélisation en 3D pour vue orthographique	Effet de fausse 2D avec des assets stylisés	Mélange de 3D et 2D (HD-2D) avec effets de caméra
Modes de jeu	Mode Versus en 1v1 Local	Mode Arcade contre une IA	Mode Campagne avec un scénario

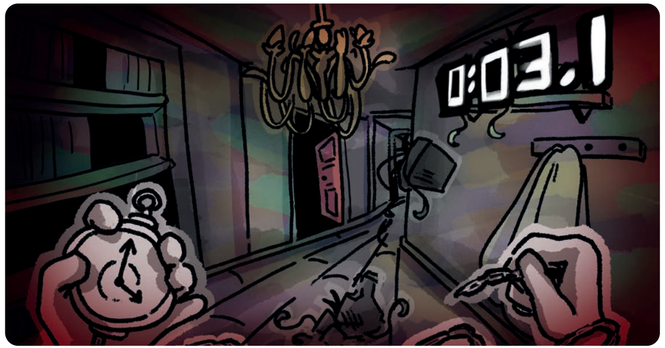
Itérations

V1 - Concept initial

Nos premières itérations se dirigeaient vers un jeu centré sur le déplacement et construit autour d'un système de balance temporelle : lorsque le compteur atteint 0, la partie redémarre automatiquement ; le joueur peut récupérer ou transmettre du temps aux objets qui l'entourent, ce qui influence directement ses capacités de déplacement. Plus son temps se rapproche de 0, plus sa vitesse augmente.

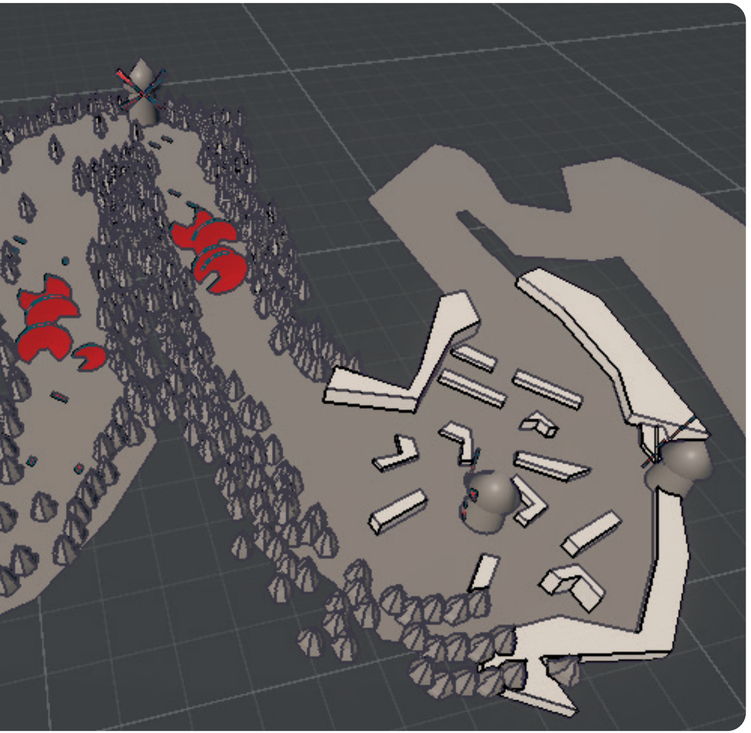
Nous cherchions à créer une forte tension à travers des phases de parkour et des prises de décision rapides dans un temps limité.

Visuellement, nous étions très inspirés de la chronophotographie, notamment grâce à l'utilisation d'images rémanentes afin de traduire l'idée de mouvement et de temporalité.



Cependant, plusieurs problèmes sont apparus au cours du développement du concept : le principe général s'est révélé trop complexe à comprendre et se rapprochait excessivement de la notion de cycle, ce qui l'éloignait du sujet initial. De plus, l'utilisation de la chronophotographie paraissait inadaptée, puisque cette technique est davantage associée à l'étude du mouvement des êtres vivants qu'à celle des objets.

Nous avons donc réorienté le projet afin de ne conserver que le principe de rembobinage temporel appliqué aux objets, tout en épurant fortement l'approche visuelle.



V2 - Toy

À partir de nos intentions de base, nous avons redirigé notre jouet autour de l'exploitation de la mécanique, notamment à travers le contrôle de trajectoires qu'elle offre.

Nous avons pris en référence des jeux comme le Golf et le Billard, où le joueur doit également réguler la trajectoire d'une balle annexe, et nous sommes demandés comment notre mécanique pouvait apporter une nouvelle expérience.

En partant de la player fantasy du « tir parfait », nous avons convenu de rediriger notre jouet autour de la régulation de la trajectoire d'une balle via la mécanique de rembobinage.

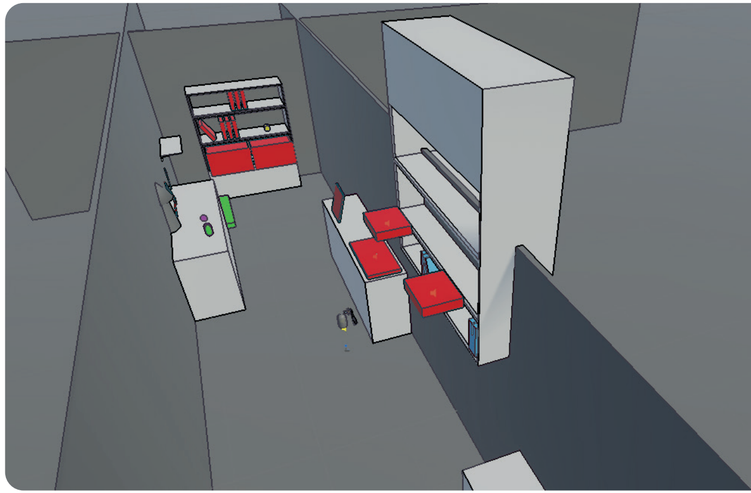
Le joueur doit alors penser les objets et les synchorniser afin de créer une trajectoire composée, à partir de chemins induits par le LD du jouet.

V3 - Premiers essais de jeu

Après avoir eu la validation de notre Toy, nous avons entamé la création de règles et d'objectifs afin d'arriver petit à petit à un jeu, en se concentrant sur l'aspect de synchronisation de notre mécanique.

Nous nous sommes orientés vers un jeu de type Puzzle, inspiré par des jeux comme Portal où le plaisir des joueurs réside dans l'itération et la manipulation de la mécanique de base, dont la compréhension constitue déjà un challenge en soi.

Nous voulions, même sous forme d'un jeu avec des objectifs, que le joueur continue à expérimenter avec la mécanique. Nos avons remarqué, lors de différentes sessions de playtests, que les joueurs appréciaient notamment manipuler les objets, que ce soit avec le Grab ou le Rewind. Nous avons fait en sorte de garder ces éléments au centre de notre gameplay. En suivant cette logique, nous avons procédé à un retravail des contrôles, de façon à améliorer le confort de manipulation du joueur en vue des puzzles à venir.



À partir de nos intentions et en étudiant des jeux similaires, nous avons conclu en la création d'un agent semi-autonome, Squeaky, autour duquel gravitent les conditions de victoire et de défaite et qui permet de concevoir des puzzles jouant principalement sur la synchronisation.

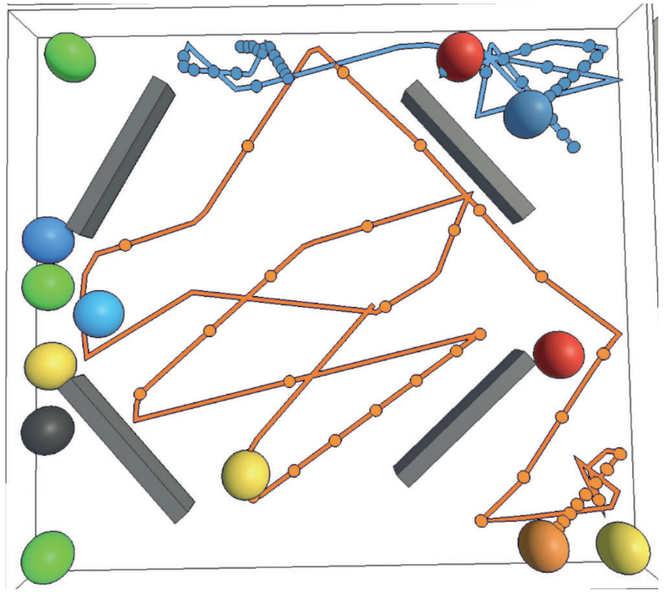
Malheureusement, cette première piste rendait la mécanique de Rewind anecdotique, le problème étant que le chaos était trop présent pour exiger du joueur une utilisation précise de la mécanique.

V4 -Transition 3D vers 2D

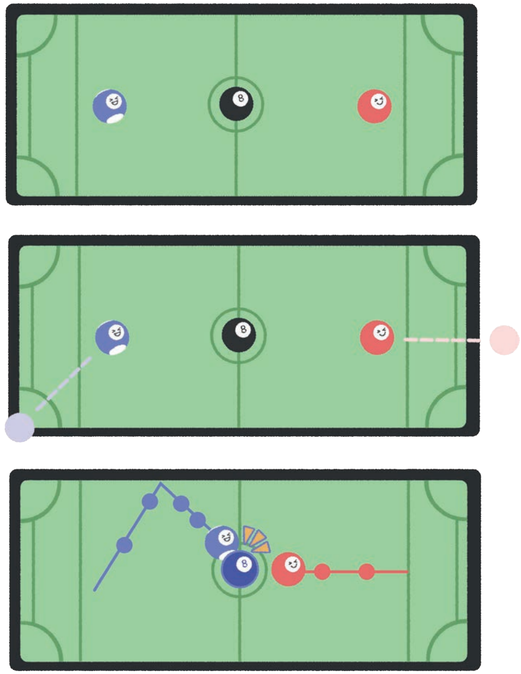
Nous avons remarqué très vite que notre mécanique de Rewind était l'exemple parfait d'un toy "Boomerang" : agréable à manipuler, mais difficile à maîtriser, rendant le passage vers le jeu compliqué.

Il était notamment impossible de provoquer des collisions dans un espace en 3 dimensions, et à partir du moment où l'on mettait dans les mains du joueur la possibilité de contrôler exactement la position spatiale des objets dans l'espace, la mécanique de Rewind devenait caduque.

Dans l'optique de réduire cette part de chaos et de reconcentrer le jeu autour de la mécanique, nous avons décidé de passer d'un espace en 3D à de la 2D top-down, ainsi que d'assumer pleinement le potentiel chaotique de notre mécanique.



C'est dans cet esprit que nous avons basculé du Puzzle Game au jeu d'arcade, avec au départ toujours l'idée d'un jeu solo. C'est également à partir de ce moment que nous nous sommes rapprochés de la thématique du billard, que ce soit en tant qu'outil de GD ou en tant qu'appui en Direction Artistique.

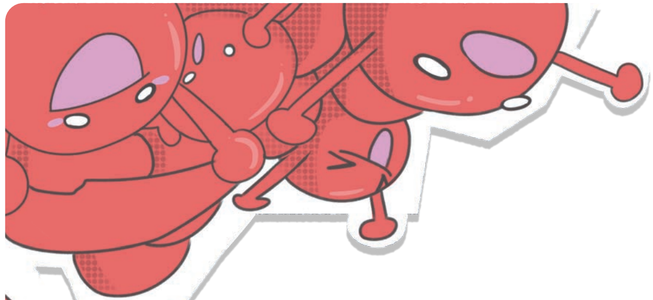


V5 : Passage au Versus et Mise en place des règles

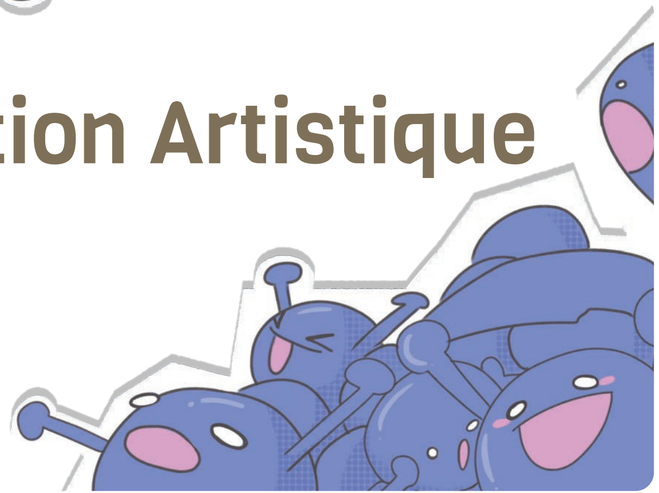
La décision de passer d'un jeu à un jeu de versus est venue lors d'expérimentations en duo sur Parsec : il y avait encore 2 slots de Rewind, et nous nous sommes mis à deux pour les utiliser, au départ dans un objectif d'entraide (réaliser une collision particulière à deux) mais basculant petit à petit vers l'opposition (empêcher le joueur adverse de réaliser une collision).

Le principal intérêt de cette façon de jouer était la réduction de la charge mentale : en n'ayant plus qu'un seul objet à Rewind par joueur, ces derniers pouvaient plus se concentrer sur leur manipulation et créer de nouvelles situations de jeu plus intéressantes.

En partant de nos expérimentations, nous avons petit à petit abouti en un jeu de versus mélangeant le principe de trajectoire et de collision du Billard ainsi que la gestion du terrain et le système de scoring du Air Hockey.



Direction Artistique



Intentions et références de départ

Le temps est un élément bien étrange : à la fois omniprésent mais invisible sans passer par des témoins de son avancée (montres, objets érodés, vieillesse), le temps a toujours été source de réflexions, qui se sont également illustrées dans les arts à travers des artistes cherchant à représenter le temps dans des oeuvres. De la même manière, tout au long du projet, nous avons fait en sorte de toujours garder, malgré l'expansion et l'étoffement de notre direction artistique, l'idée de **représentation graphique du temps**.

Nos premières pistes graphiques se sont immédiatement rapprochées du principe de chronophotographie, qui consiste en la représentation imagée d'un mouvement dans l'espace à travers la **combinaison de plusieurs clichés**.

De façon plus générale, nous avions très tôt identifié un **intérêt graphique** à la représentation d'un objet à travers ses états successifs dans le temps : au-delà de l'aspect surnaturel de telles images, invisibles dans la nature et uniquement reproduisibles à l'aide d'outils, il y a toute une idée de **trajectoire** qui se crée dans le temps que nous trouvons pertinente pour notre mécanique.

***The Entropy Centre**, Stubby Games, 2022*

On retrouve déjà dans des jeux la représentation graphique du mouvement d'un objet : d'ordinaire courte et utilisée comme un feedback temporaire, des jeux comme ***The Legend of Zelda : Tears of The Kingdom*** et ***The Entropy Centre*** sont des références quant à la représentation graphique continue d'un objet dans le temps et l'espace. Ce sont ces deux principales références qui nous ont guidées en début de projet dans la manière de représenter un objet et son déplacement passé dans l'espace.

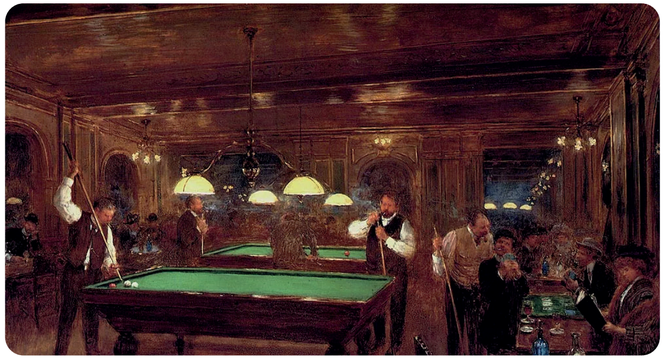
Thématique du Billard

D'abord issue d'une association d'après nos mécaniques de Game Design, la thématique du billard est quant à elle très vite devenue un tremplin pour la construction de l'univers graphique du jeu, dans la mesure où elle permettait de mettre en commun notre concept de jeu axé sur les collisions et les trajectoires avec un sport populaire dans la même veine.

D'ordinaire, le billard est un jeu de précision séquencé, où les joueurs jouent les uns après les autres : on retrouve dans des œuvres comme ***Le Billard*** de Jean-Georges Beraud, l'association du billard à des joueurs expérimentés, avec cette idée d'un jeu de réflexion, accompagnés de spectateurs.

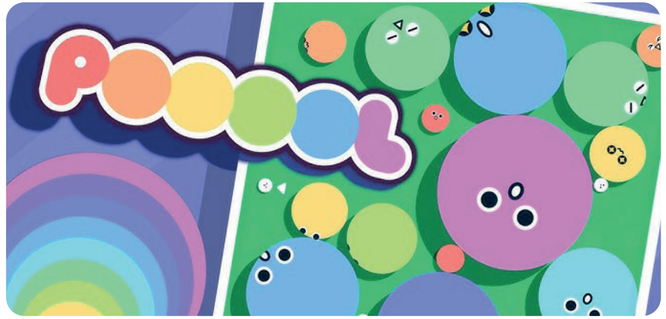
Le billard est surtout un sport **regardé** et créant de l'**engouement** par son **rendu visuel** : les trajectoires des boules au billard impressionent, et l'on retrouve une certaine satifcation à la réalisation de coups maîtrisés.

Cette idée en particulier s'associe avec nos intentions pour notre jeu, avec une mécanique de retour dans le temps qui produit ce même type de satisfaction.



*Jean Georges Béraud, **Le Billard,** XXe*

***Subpar Pool**, Grapefrukt, 2023*



Des jeux pré-existants ont déjà pu utiliser le thème du billard en l'associant à leurs mécaniques :

- Dans **Subpar Pool**, on retrouve l'idée de créer des trajectoires avec une boule principale, dans l'objectif d'amener des boules secondaires à des endroits clés de la zone de jeu ;
- Dans **Poool**, on retrouve l'idée de collisions fréquentes entre des boules de différentes couleurs, pouvant rappeler la casse (ouverture de la partie avec un premier coup servant à répartir les boules dans l'espace de jeu).

Dans ces deux jeux, le billard est notamment utilisé pour faciliter l'association d'éléments mécaniques avec des éléments de jeu du billard déjà connus du joueur : cela permet de faciliter l'**assimilation** de certaines règles du jeu.

***POOOOL**, Noah King, 2024*

Modernisation thématique

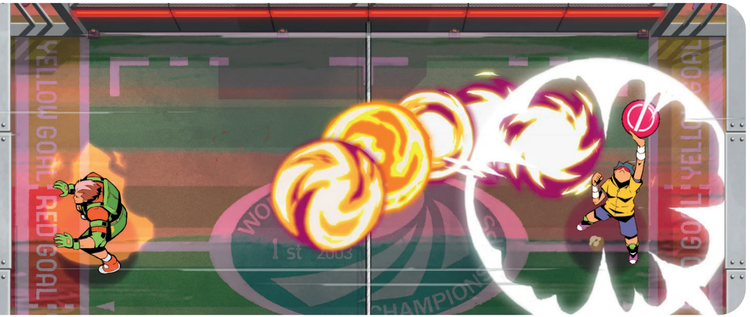
Néanmoins, le thème du billard seul dans son essence provoque un contresens quant à nos intentions de jeu : là où notre jeu se veut **réactif, dynamique et convivial**, en visant un public large sans besoin immédiat de maîtrise poussée de la mécanique, le billard est d'ordinaire un sport **codifié, complexe et technique**, mais surtout **séquencé et lent** dans son exécution.

Cette opposition présente, il a été nécessaire de transformer ce thème initial en vue de se rapprocher de nos intentions d'expérience de jeu.

Ce travail de transformation a été référencé grâce à différents jeux ayant déjà effectué ce travail de transformation d'un sport pré-existant : **Beyblade**, **Inazuma Eleven**, **Windjammers** ou encore **Super Mario Strikers**.

***Beyblade X Xone**, FuRyu, 2024*

***Mario Strikers : Battle League Football**, Nintendo, 2022*



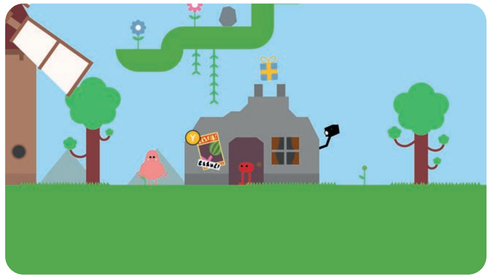
***Windjammers 2**, Dotemu, 2022*

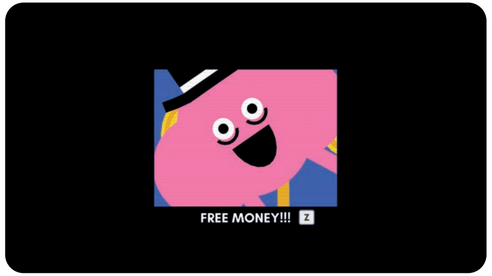
***Inazuma Eleven**, Level-5, 2008*

Dans ces références, on retrouve l'utilisation de procédés graphiques amplifiés dans les **VFX** et tout un travail de **Narrative Design**, qui permettent de transformer un sport existant dans une expérience radicalement différente se prêtant plus au jeu vidéo.

En se basant sur ces références, nous avons pu construire notre jeu autour de l'idée d'un **climax d'une partie de billard**, où deux joueurs se battent pour emporter la dernière boule n°8 et transcendent les règles du temps : ce procédé in medias res permet de directement placer les joueurs dans un instant à haut enjeux, qui convient plus au type d'expérience proposé par le jeu.

Enfin, nous plaçons notre utilisation du billard dans une envie de créer une **expérience décalée**, à travers le contraste entre un sport calme devenant dynamique dans un exemple de partie à haute tension.





***Pikuniku**, Sectordub, 2019*

Stylisation

Notre procédé de stylisation graphique s'inscrit entre deux besoin principaux : la lisibilité, nécessaire au déploiement du Rewind, ainsi que le dynamisme, rattaché à nos intentions d'expérience. Dans cet esprit, nous avons combiné deux types d'esthétiques afin d'aboutir à un entre-deux.

Minimalisme

L'esthétique Minimaliste n'est pas à utiliser en terme de nombre d'éléments de jeu, mais plutôt quant à la **complexité des formes présentes**.

Notre principale référence a été ***Pikuniku***, qui correspondait à notre besoin du minimalisme : cela permet d'assurer la lisibilité, avec des formes simples ou composées à partir de cercles notamment. C'est également une référence qui montre que le **minimaliste peut être utilisé pour des univers se voulant décalés**.

Enfin, le minimalisme permet, à travers ses formes courbes, de rendre le jeu **visuellement abordable** auprès de joueurs plus casuals : la plupart des jeux de combat, avec leur aspect plutôt agressif (souvent provoqué par l'utilisation de lignes noires fortes ou de style réaliste), intimident les casuals ; à travers notre utilisation du minimalisme, nous voulions surtout éviter ce biais, en créant une ambiance plus accessible ouverte à tous.

Esthétique Pop et héritages de la sérigraphie

L'esthétique Pop quant à elle nous sert de référence pour tout l'apport dynamique : on retrouve dans les oeuvres de Roy Lichtenstein et de façon générale dans les comics des années 60, l'utilisation de **formes extravagantes** et de **couleurs variées** pour créer des effets de mouvement et de dynamisme ; l'artiste Keith Haring est notamment une référence quant à l'association du Pop Art avec une approche plus **minimaliste**.

*Keith Haring, **Untitled,** 1988*

*Roy Lichtenstein, **Explosion,** 1967*

Graphiquement, le Pop Art et plus précisément la pratique de la **sérigraphie** par des artistes comme Andy Warhol nous a permis de transformer nos formes vectorielles de base afin qu'elles soient plus vivantes et moins lisses : en s'inspirant de l'**effet de texture** produit par les bavures d'encre lors des impressions, nous avons appliqué un procédé similaire à nos assets, les rendant ainsi plus **texturés et vivants**.

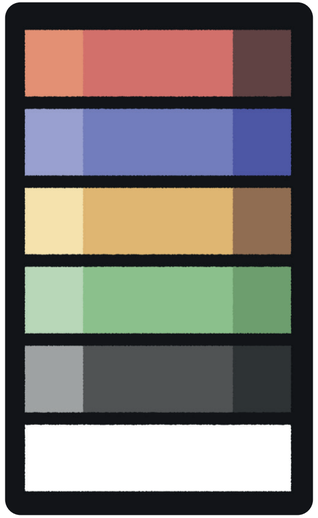
Enfin, il a été question pendant un moment de s'inspirer des **trames** pour notre jeu : cependant, cela amenait une **complexification trop grande** des éléments de jeu, et ce procédé a été gardé uniquement pour des **assets plus secondaires** pouvant se permettre cette complexification. Nous avons tout de même voulu faire un écho à ce procédé à travers l'utilisation de textures de points dans différents éléments de jeu comme le fond.

*Andy Warhol, **Liz,** 1964*

Couleurs

La palette de couleur, en plus d'être inspirée par le pop art et les teintes des oeuvres sérigraphiées d'époque, se veut également **très codifiée** : chaque élément de jeu est associé à une couleur, et en fonction du degré d'importance du feedback, la couleur sera plus ou moins saturée.

Cette manière de procéder, inspirée par des jeux comme ***Baba is You***, permet d'avoir des feedbacks codifiées mieux interprétables périphériquement : dans un jeu de Versus très mouvementé où le joueur bouge constamment son regard, nous trouvions important qu'il puisse identifier un feedback dans son **champ de vision** sans avoir à se focaliser dessus.



Enfin, nous avons choisi de ne pas appliquer d'effet d'**ombres ou de lumière** directement sur les objets, de façon à garder leur apparence simple. Néanmoins, nous avons tout de même utilisé des effets d'ombres pour créer un **effet de superposition** des éléments de jeu entre eux, afin d'apporter plus de profondeur.

*Andy Warhol, **A Set of Six Self-Portrait,** 1967*

***Baba is You,** Hempuli, 2019*

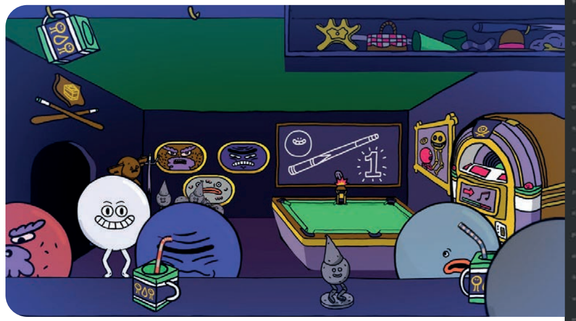
Boules

Pour les boules des joueurs, le principal enjeu était de réussir à donner envie au joueur de s'y attacher : c'est à cet effet que nous avons utilisé le procédé de personnification à travers l'ajout d'un petit visage aux deux boules.

Ce visage change d'émotion en fonction des actions dans le jeu (collisions, impulse, goal), ce qui permet de retranscrire l'idée que les boules ont des émotions et réagissent aux évènements en jeu.

Nos principales références quant à cette personnification ont été **Pool Panic** et **Peak**, notamment afin d'avoir des idées quant aux types d'expressions faciales et quant à la place du visage sur la boule de base.

Afin de différencier les deux boules des deux joueurs, nous nous sommes permis un éloignement du billard : là où les deux boules auraient dû être rouge bordeaux dans une partie normale, nous avons pris la décision de les différencier respectivement en rouge plus saturé et en bleu ; néanmoins, nous nous rapprochons tout de même du billard traditionnel en ayant la distinction pleine et rayée.



***Pool Panic**, Rekim, 2018*

***Peak**, Aggro Crab, 2025*

Sprites des deux boules joueur (2D/UI)

Texturing des deux boules joueur (3D)

8-Ball

Pour la 8-Ball, il était nécessaire de la différencier des deux boules joueur : cette différenciation se fait grâce à leur différence de couleur, avec une 8-Ball grise foncé ; de plus, sa couleur change de teinte (mais pas de saturation) en fonction du dernier joueur ayant été en contact avec.

Nous avons également décidé de conserver le numéro et de ne pas lui donner de visage : cela permet de distinguer les boules joueurs (animées et expressives) de la 8-Ball (objectif de jeu non-incarné).

Texturing

Les textures des boules ont été réalisées sur Procreate : c'était l'occasion de s'essayer à une autre pipeline sur un logiciel d'habitude réservé à de la 2D, et procéder ainsi nous a permis de pouvoir utiliser des brush de Procreate se rapprochant de nos intentions de style graphique.

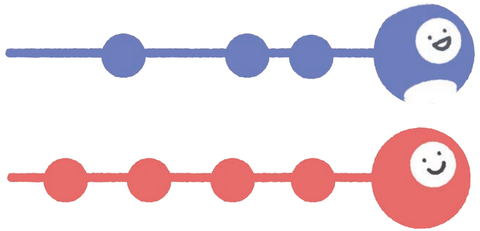
Feedbacks

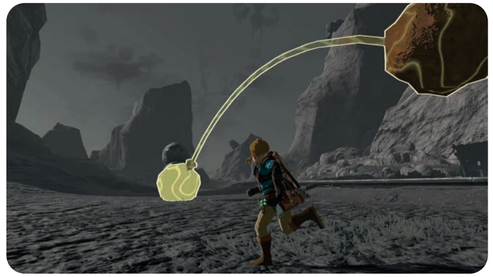
Rewind

La trail du Rewind a principalement été inspirée par nos références de départ pour la mécanique temporelle : nous avons notamment repris le système d'outline de **TOTK** afin de faciliter la séparation graphique du reste des éléments de jeu.

Nous avons également mis en place un système de points, géré par script, qui sert de feedback quant à la vitesse de l'objet dans ses sauvegardes précédentes : de la même manière que pour la chronophotographie, plus un objet est lent, plus ses images seront rapprochées entre elles.

De la même façon, la présence de têtes sur les deux boules joueur sert de repère quant au retour temporel des objets : cela permet de ne pas avoir l'impression d'un objet sur un rail, mais bien d'un objet avec sa rotation propre qui s'inverse lorsqu'il retourne dans le passé.





The Legend of Zelda : Tears of the Kingdom, *Nintendo, 2023*

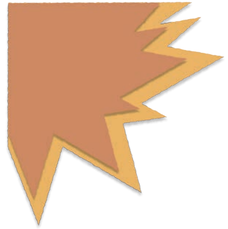
Richard Fauguet, ***Sans Titre,*** *2004*

Impulse

L'Impulse possède deux feedbacks successifs :

- Une flèche indicatrice apparaît dès que le joystick est actionné, peu importe si le joueur est en plein cooldown ou non. Ce feedback contextuel, inspiré par celui de **Curse of the Dead Gods**, permet à la fois d'indiquer la direction de visée de l'Impulse, son cooldown et sa puissance de charge ;
- Un effet d'impact ainsi qu'une bulle apparaissent dans l'UI à côté de la tête du joueur dès que ce dernier est actionné : ces feedbacks sont plus stylisés, en s'inspirant notamment de ceux présents dans **The World Ends With You**, et servent à la fois à renforcer le dynamisme du jeu (avec notamment l'idée d'un personnage qui crie son coup spécial), mais aussi de feedback pour le joueur adverse en l'informant des actions de son adversaire.

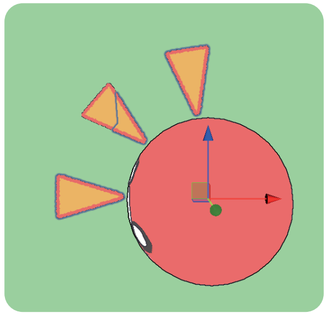
Nous aurions également aimé pouvoir appliquer sur la boule un effet de flammes graduel en fonction de la force d'impulsion mais le temps nous a manqué, cet ajout nécessitant de nombreuses itérations pour assurer la lisibilité en jeu.



The World Ends With You - Final Remix*, Square Enix, 2016*

Curse of the Dead Gods*,*
Passtech Games, 2020

Keith Haring, ***Untitled Dance,*** *1987*



Collisions

La collision est indiquée en jeu par deux feedbacks :

- L'expression du visage de la boule joueur, dans l'UI comme sur le modèle 3D de la boule, changent dès que le joueur subit une collision : ce changement permet de participer à la personnification des avatars ;
- Des particules stylisées apparaissent autour du joueur, orientées vers le point de contact : ce feedback est inspiré par les VFX issus du Pop Art et notamment ceux que Keith Haring, qui utilise régulièrement des groupes de 3 petites lignes pour signifier le mouvement dans ses oeuvres.

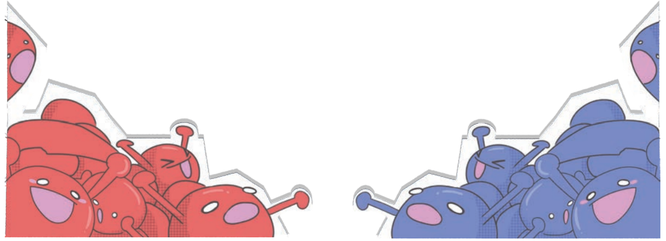
Les particules de collision ont notamment été réalisées grâce au Particle System d'Unity, qui permet d'instancier un nombre précis de particules à partir d'un arc placé autour du joueur : c'est cet arc que l'on va tourner en Y pour l'orienter vers le point de collision. On décide ensuite via le mode d'émission Burst de disperser 3 particules à équidistance sur cet axe.

Ambiances

Certains feedbacks servent plus d'ambiances que de réels feedbacks utiles pour les joueurs en jeu :

- les Supporters, qui apparaissent dès qu'un joueur marquent un point, sont inspirées du public de **Paper Mario et la Porte Millénaire** et servent de récompense pour le joueur victorieux.
- le Commentateur, qui change de dialogue en fonction des derniers évènements en jeu (collision, point remporté, ect...) est quant à lui inspiré du navigateur de **Persona 4 Arena Ultimax**, qui commente des actions en jeu.

Ces deux feedbacks sont particuliers car ne servant pas directement et uniquement à rapporter l'évolution de l'état système aux joueurs : le premier sert de récompense visuelle, tandis que le second sert de feedback plutôt pour les spectateurs, qui peuvent ainsi suivre les différents évènements en jeu à la manière d'un commentateur.



Persona 4 Arena Ultima*, Atlus, 2013*

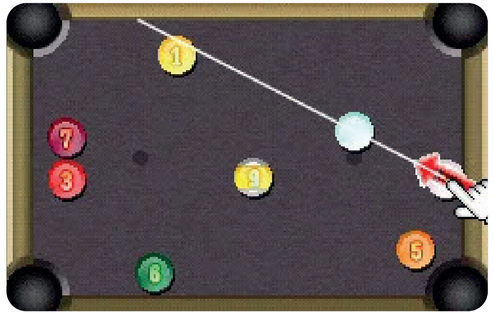
Dans les deux cas, ce sont des feedbacks qui apportent énormément à l'ambiance et l'aspect vivant du jeu, en rappelant l'idée d'un sport regardé qui crée de l'engouement.

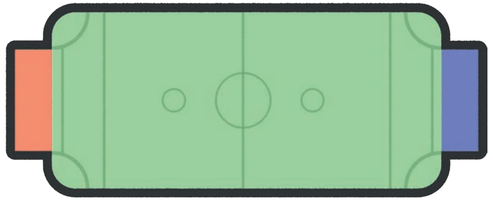
Terrain de jeu

Le terrain correspond à un mélange entre une table traditionnelle de billard et des éléments graphiques issus du Air Hockey :

- On retient du billard la forme générale de la table ainsi que la couleur du tissu verte qui permet de démarquer les éléments situés dessus ;
- Pour le Air Hockey, on retient notamment les différents marquages et lignes qui permettent de différencier visuellement les camps des deux joueurs.

Le terrain est visible dans son intégralité à tout moment de jeu, de la même manière que dans des jeux comme **Windjammers** et **Pong** : nous voulions donner au joueur une vue globale sur la situation de jeu lui permettant de pouvoir planifier son plan de jeu en fonction de son état de jeu et de celui de son adversaire.





Background

Le fond du jeu se veut assez simple, dans la continuité de nos besoins de lisibilité et afin de mettre en avant les principaux éléments de jeu (notamment le terrain).

Nous avons opté pour un fond uni, accompagné d'un pattern à pois défilant en diagonale : ce défilement permet d'ajouter malgré sa simplicité du mouvement au fond, participant à la construction de l'ambiance dynamique du jeu.

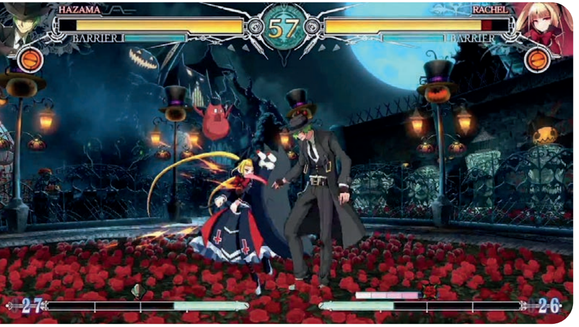
Nos références pour l'exécution de ce fond sont **Puyo Puyo Tetris**, notamment pour la taille du pattern à pois, ainsi que **ChuChu Rocket** pour la vitesse de défilement du pattern.

Ce défilement a été réalisé dans Unity à partir d'un cylindre que l'on fait tourner en jeu.

Puyo Puyo Tetris, *Sonic Team, 2014*

ChuChu Rocket, *Sonic Team, 1999*

***Windjammers 2**, Dotemu, 2022*



***Blazblue : Central Fiction**,*
Arc System Works, 2015

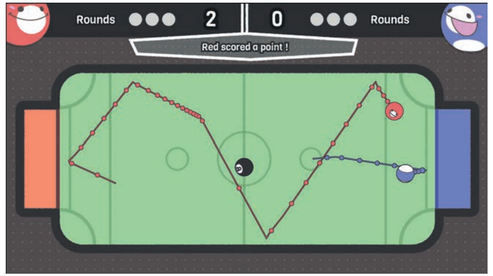
UI

L'UI a également été pensée dans la continuité de notre idée de feedbacks périphériques : le principal enjeu était de trouver un juste équilibre entre lisibilité et identité, notamment à travers la place prise par les différents éléments à l'écran.

Après de nombreuses itérations, nous avons trouvé un juste équilibre permettant à la fois d'avoir un terrain d'assez grande taille pour être lisible ainsi que différents feedbacks périphériques qui contribuent à l'ambiance du jeu.

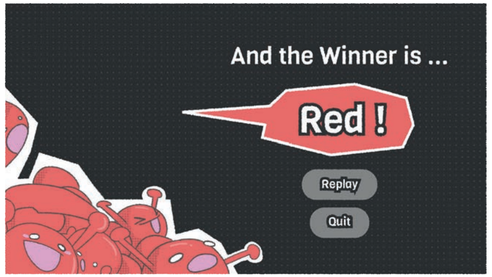
L'écran de jeu principal et les éléments le composant sont principalement inspirés de jeux de combats comme **Windjammers** ou encore **Blazblue**, avec la concentration des informations de jeu en haut de l'écran et la séparation de l'écran en deux pour chaque joueur.

Les différents éléments ont été réalisés en vectoriel sur Figma, agencés dans des mockups pour être ensuite importés dans Unity : cette façon de procéder permet notamment de pouvoir vérifier la taille des éléments, mais aussi d'appliquer un effet d'ombre créant une sensation de profondeur et de superposition des différents élements graphiques.



Ecran de jeu en partie

Ecran Titre



Ecrans de victoire

Logo

Pour le logo, l'enjeu principal était de retranscrire d'un coup d'oeil les principaux éléments du jeu : la mécanique de Rewind, la thématique du billard et l'aspect Versus du jeu. Le nom **8 o'Clock** est venu en mélangeant le "8 Pool", nom d'une variante américaine du billard, avec "O'Clock", contraction anglophone utilisée pour parler des heures : cette contraction permet de rappeler le lien avec le temps et le billard.

Le logo a été réalisé sur Figma en vectoriel dans le même esprit : en partant de la police Varela Round, les différents éléments ont été construits autour petit à petit. On retrouve notamment un mélange entre la rondeur des lettres et des boules avec les côtés plus tranchants du contour et des FX d'impact : l'idée était de représenter à la fois le côté sans prise de tête et loufoque du jeu mélangé à son esprit compétitif et l'impact des collisions.

***Windjammers 2**, Dotemu, 2022*

Anciennes itérations

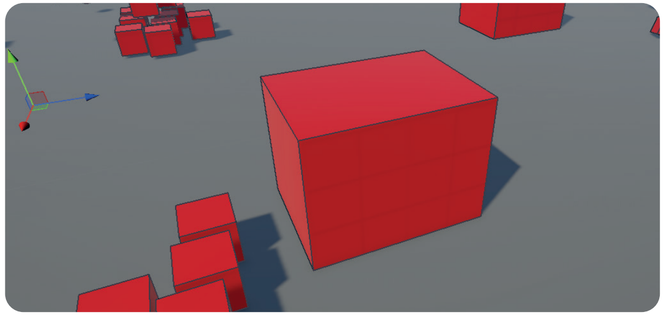
Shaders

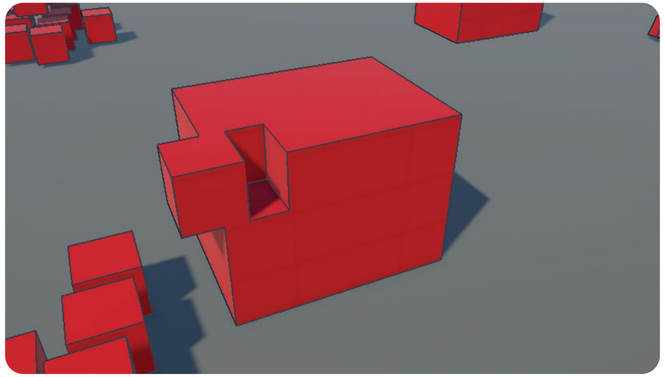
Nous avons choisi d'utiliser un shader d'outline pour permettre de faire ressortir les objets de la scène et leur donner un effet dessin : ce shader de post-traitement fullscreen fonctionne sur une image déjà rendue de Unity ; il n'a donc aucun impact sur la géométrie des objets présents dans la scène.

Son fonctionnement est le même que celui d'un œil humain : il analyse un changement de couleur brutal en prenant un pixel et en analysant ceux à côté de lui ; cette méthode s'appelle le filtre de Robert Cross.

Pour chaque pixel, le shader regarde ses quatre pixels voisins (haut, gauche, droite, bas) : normalement, le filtre Robert Cross se base sur des diagonales, mais ici, son fonctionnement est simplifié.

Il soustrait la valeur du pixel de gauche à celui de droite (de même verticalement), puis mesure l'amplitude du résultat : si l'amplitude dépasse le seuil alors le pixel est considéré comme un bord d'objet.





Cependant, cela n'est pas suffisant seul : il regarde aussi les normales de surface des objets, c'est-à-dire les directions de chaque face d'un objet.

Pour ce qui est des normales, elles sont stockées dans le buffer de normales et fonctionne aussi comme le filtre Robert Cross : le buffer stocke pour chaque pixel la direction de la surface de l'objet. Quand deux surfaces forment une arête, les normales changent d'un pixel à l'autre ce qui permet d'appliquer le shader comme pour le changement de couleur.

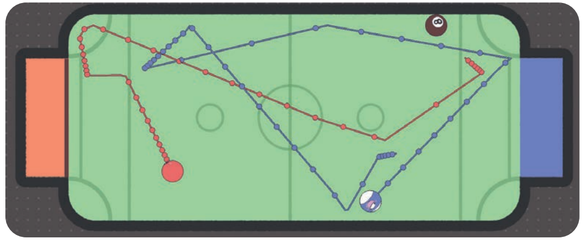
Si 2 objets, de la même couleur et taille sont collés, alors le shader prendra la forme complète de l'objet. Le shader se base sur les normales de surface pour détecter tout de même l'arête entre eux mais sans l'afficher car leurs normales sont identiques.

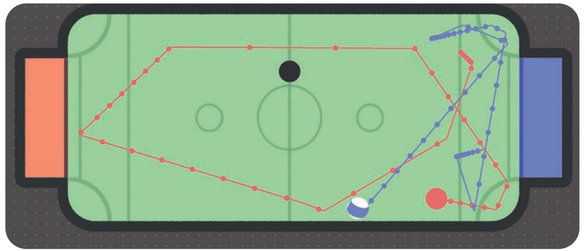
Utilisation

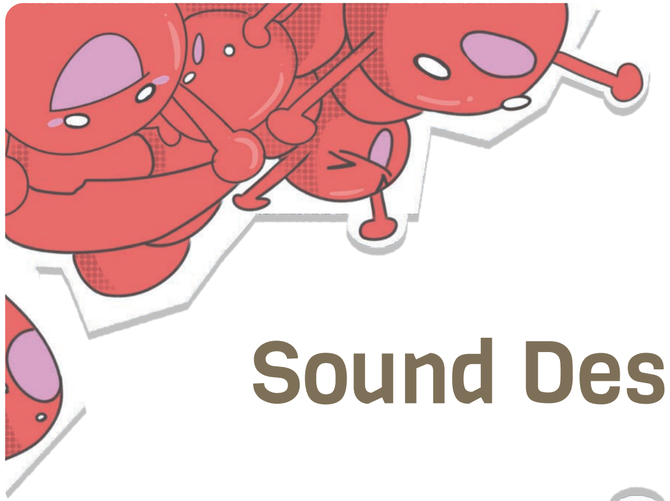
Nous avons choisi d'utiliser un shader d'outline pour améliorer la lisibilité des objets interactifs dans notre scène.

Notre environnement est composé de formes géométriques simples avec une palette de couleur saturée. Il est essentiel pour nous que le joueur puisse identifier du premier coup les objets avec lesquels il va pouvoir interagir pour améliorer son expérience de jeu.

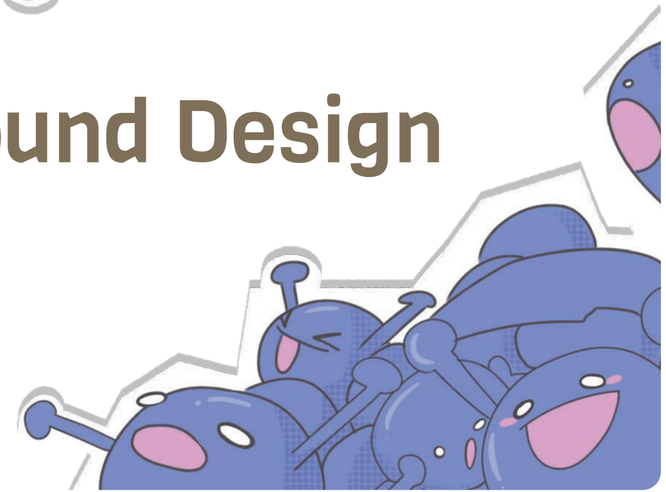
Notre jeu tend vers un espace de jeu saturé via les trails créées mais aussi les mouvements de boules dans tous les sens : les trails se superposent, se croisent et il est difficile à un certain moment de comprendre ce qu'il se passe sur l'écran.







Sound Design



Intentions et références

L'identité sonore du jeu repose sur une volonté claire : accompagner et amplifier le plaisir de jeu sans jamais déranger le joueur.

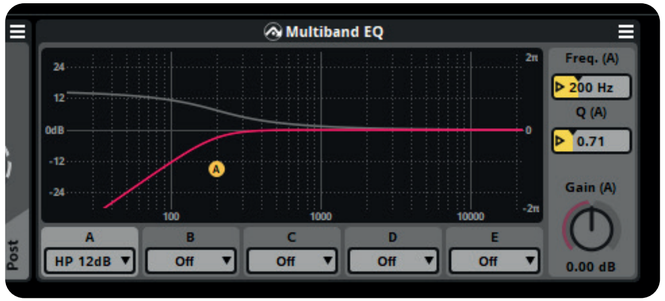
En cohérence avec une direction artistique pop, le sound design se veut percutant, dynamique et énergisant, avec l'idée que chaque son doit avoir un impact immédiat et participer à la satisfaction du joueur.

L'objectif est de créer un environnement sonore qui pousse le joueur à continuer, à la manière des grands jeux d'arène et de sport : on entend les collisions, ressent les buts et on perçoit l'énergie du terrain.

Références principales

- **Rocket League** : Impact des collisions, les sons de score satisfaisants et l'énergie générale du sound design sportif ;
- **Super Smash Bros** : musicalité des coups, rythme et lisibilité sonore des actions ;
- **Street FIghters** : Atmosphère tendue et compétitive, montée en intensité ;
- **Windjammers 2** : Musique principale.

Réalisation des sons



Sons de feedbacks et d'interface

Plutôt que d'utiliser des sons «attendus» (un choc pour une collision, une sirène pour un but), le parti pris a été de partir de matières sonores détournées issues de banques de sons libres de droits avec des sons qui, à l'origine, n'avaient aucun rapport avec leur usage final.

Ces éléments ont ensuite été travaillés sur Reaper : découpe, layering, pitch shifting, compression, EQ, afin de créer des sons qui correspondent à l'esthétique pop et énergique du jeu.

Ce processus permet d'obtenir un sound design original, cohérent et reconnaissable qui place le jeu dans cet esprit pop et décalé.

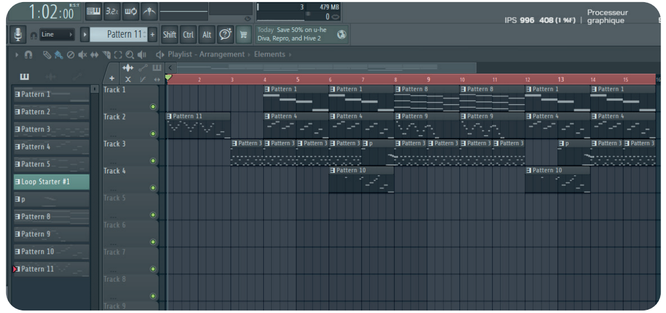
Musique originale

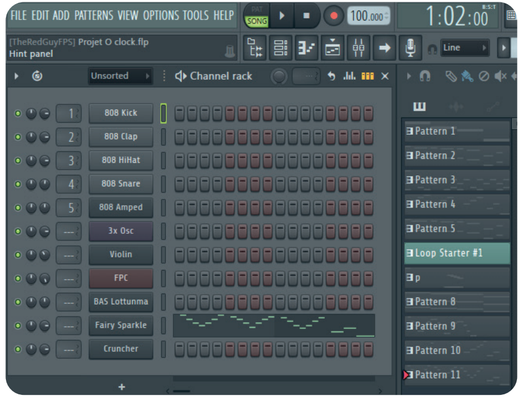
La musique du jeu a été entièrement composée sur FL Studio.

La composition intègre des lignes de basse, des percussions rythmées et des mélodies synthétiques inspirées de l'esthétique des années 80-90 revisitée.

La référence principale est le thème de sélection de personnage de **Windjammers 2**.

L'objectif était de produire une musique qui s'efface quand il le faut mais qui booste l'adrénaline aux moments clés.



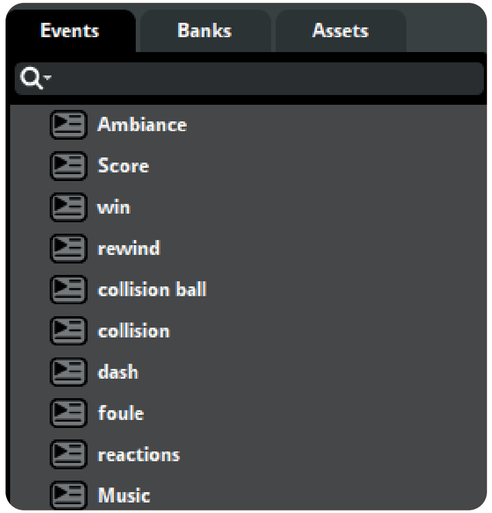


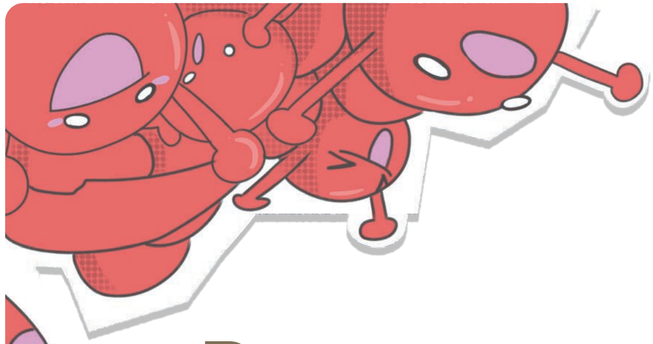
Intégration dans FMOD

L'ensemble du sound design a été intégré via FMOD Studio, qui assure le pont entre les assets audio et le moteur de jeu.

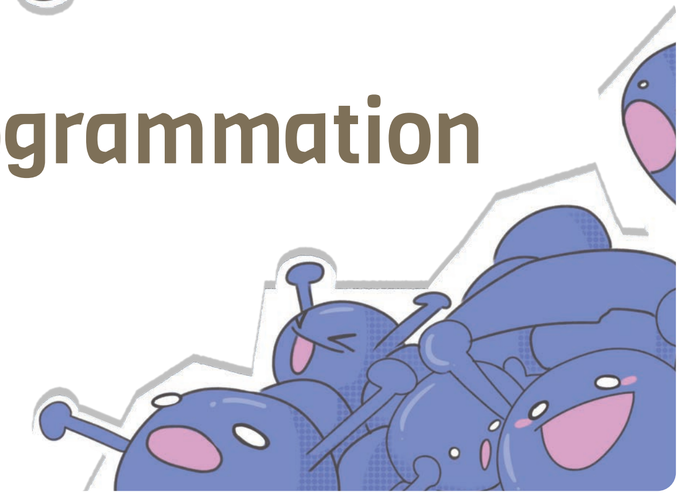
Chaque événement sonore a été configuré pour être déclenché au bon moment, dans le bon contexte :

- les sons de gameplay (collisions, buts, score) sont appelés en temps réel en réponse aux événements physiques du jeu ;
- les sons de menus sont gérés de façon indépendante, avec leurs propres transitions ;
- la musique est gérée comme un événement continu avec gestion du volume et des transitions entre les états du jeu ;
- l'ambiance sonore est jouée en couche de fond, ajustable selon l'intensité de la partie.





Programmation



Introduction

La programmation pour 8 o'clock fut un véritable challenge : nous sommes passés par un nombre conséquent de reboots qui ont mené notre mécanique principale, le rewind, à être constamment transformée pour suivre chaque nouvelle direction.

Plusieurs problématiques ont émergé dont l'infiltration de code rémanent d'anciennes versions devenues obsolètes dans les nouvelles versions : toutes ces problématiques ont mené à un *refacto* de plusieurs scripts dont celui du joueur ce qui sera plus amplement expliqué plus tard.

Nous commencerons par parler des enjeux techniques et des problématiques évidentes à résoudre, puis nous parlerons de tous les éléments qui constituent une boule joueur et donc de la mécanique principale.

Nous parlerons ensuite de la gestion, filtrage et interprétation des inputs ; enfin, nous verrons le système de score et de la boule 8. Nous survolerons également le fonctionnement et l'évolution du rewind, en évoquant les modifications principales par version.

Enjeux techniques

L'idée de retour dans le temps fut évoquée très tôt : plusieurs chemins s'ouvraient à nous ; néanmoins nous avions pour philosophie de créer une mécanique émergente, une sorte d'outil avec lequel on pourrait construire. Cela signifiait donc que les idées théatralisées (cad nécessitant d'être prévu par script), tel que rajeunir un objet ou une scène, furent très vite abandonnées.

L'idée la plus réaliste et faisable pour respecter notre philosophie de design s'est donc révélée être le retour dans le temps à court terme d'objets : on parle d'une échelle de temps très courte, pas plus de 10 secondes, que le joueur peut vivre en somme.

Une si petite échelle de temps permet, en temps normal, de ne remarquer que des changements physiques instantanés : un changement de position, de vitesse, ect...

Avec ces principes cœur, il est devenu évident qu'une façon d'enregistrer des données, les ranger et les réutiliser devait être mise en place de manière efficace et optimisée.

De plus, il fallait aussi faire en sorte que le retour dans le temps soit plus qu'une "annulation" d'un événement passé, mais bien une inversion qui elle-même peut avoir un impact sur le présent autre que l'objet affecté par le rewind. Il fallait donc que cette mécanique s'inscrive dans la physique du jeu.

Enfin, il était réellement important que les feedbacks soient clairs quant aux informations enregistrées, car notre mécanique n'est pas aussi simple et évidente à utiliser : il fallait donc trouver une façon d'indiquer les trajectoires passées et la vitesse d'un objet à un moment T.

Par rapport à cet aspect, l'un des principaux enjeux fut l'optimisation : sur les premières versions, on ne parlait pas encore de deux joueurs pouvant rewind, mais plutôt d'objets qu'un joueur solo peut rewind, ce qui signifiait qu'on pouvait avoir un grand nombre d'objets, tous avec leurs propres enregistrements (et feedbacks !).

I. La boule joueur

Tout d'abord, il est important de préciser que l'aspect multijoueur est atteint grâce à l'**Input Manager** et le composant **Player Input,** fournis par l'environnement de travail Unity :c'est un outil assez puissant et très rapide d'utilisation pour une implémentation rapide du multijoueur local.

La boule joueur est une sphère simple : elle possède un Rigidbody et plusieurs scripts.

Notre boule joueur est plus lourde que la boule 8 qui ne fait que 0,1 kg, car il faut que le joueur soit capable de facilement projeter la boule 8 au contact. Elle rebondit également pour permettre plus de trajectoires intéressantes.

Caractéristiques physiques

- 1 kg
- Physics Material rebondissant

A.) Le PlayerScript - La mécanique principale

La boule joueur est constituée de plusieurs scripts dont un central : **PlayerScript.**

Ce script gérait initialement uniquement la mécanique principale : plus tard, pour avoir un code plus lisible et simple à refactoriser, la majorité des mécaniques ont été centralisées dans ce script et découpées en plusieurs méthodes nommées pour être facilement lisible les unes après les autres.

Le **PlayerScript** gère donc :

- L'enregistrement des données **ChronoData** ;
- Le Rewind ;
- Les feedbacks d'enregistrement et de rewind ;
- L'impulsion de la boule ;
- Le filtrage et l'interprétation des inputs du joueur.

Tous ces concepts vont maintenant être expliqués en détail.

a.) Découpage du temps et rangement.

La première question qu'on va se poser est celle-ci :

Quelles sont les informations qu'on peut extraire d'un objet à un moment précis dans le temps ?

Dans notre vision du retour dans le temps, on peut extraire ces informations :

- La position ;
- La rotation ;
- La vélocité linéaire ;
- La vélocité angulaire.

Et c'est tout ! Ce sont les seules informations dont on a besoin pour reconstituer un mouvement.

On peut donc déjà former notre **struct ChronoData**.

```
Frequently called  2 usages  BartProjectCode
public ChronoData(Vector3 position, Quaternion rotation, Vector3 velocity, Vector3 rotationVelocity, int iteration)
{
    currentPosition = position;
    currentRotation = rotation;
    currentVelocity = velocity;
    currentAngularVelocity = rotationVelocity;
    iterationCount = iteration;
```


Le struct **ChronoData** représente donc un format de sauvegarde temporelle d'un objet physique : une itération de **ChronoData** est appelée **chronoData**.

Pour enregistrer un **chronoData** on utilise la méthode **CreateChronoData()** qui a pour fonction de :

- créer une sauvegarde ;
- l'ajouter à une liste de **chronoData** ;
- réajuster les feedbacks d'enregistrement.

Dans la vie, on peut considérer que le temps est découpable en une infinité de parts, soit une infinité de **chronoData**. Bien sûr, pour créer l'illusion, nous n'avons pas besoin d'autant de parts.

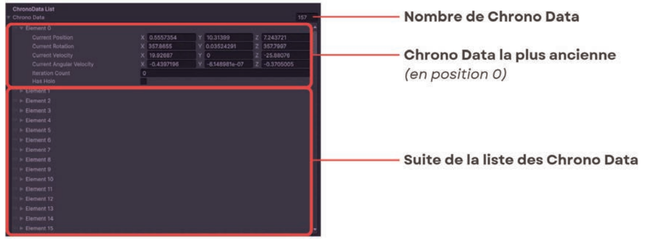
Dans le cinéma, la fréquence d'image par seconde est habituellement de 24, soit une fréquence de **24Hz**. C'est une fréquence largement suffisante pour que notre cerveau reconstitue un mouvement lorsqu'on enchaîne ces images, mais est souvent trop peu pour des médias interactifs tel que le jeu vidéo.

Bien sûr, **plus la fréquence est haute, plus l'enregistrement est précis et fluide.** Il est aussi important de garder une **fréquence stable** tout le long de l'enregistrement, sans quoi, il peut paraître saccadé et rendre le visionnage très désagréable, surtout dans les jeux vidéo.

Pour garder une fréquence stable, on utilise la **méthode FixedUpdate() qui aura le même nombre de d'appels d'une machine à une autre**.

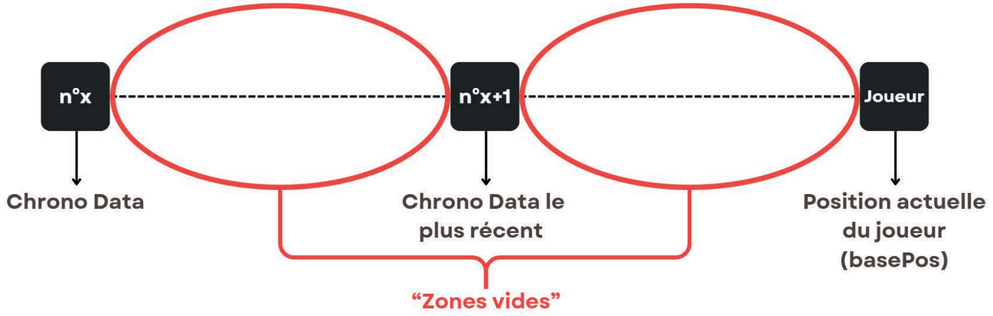
Par défaut, **FixedUpdate()**, est appelé **50 fois par seconde**, dans notre cas, nous avons multiplié par deux le nombre d'appels, élevant donc le nombre à **100**. Nous avons des collisions plus précises et d'autres avantages que nous expliquerons plus tard.

CreateChronoData() est donc appelé dans **FixedUpdate()** une fois sur deux, nous permettant d'avoir 50 chronoData par seconde. Ici nous ne faisons pas de chronoData 100 fois par secondes car en avoir plus de 50 n'est pas nécessaire.



b.) Le Rewind ou comment passer l'état actuel de l'objet à un état sauvegardé dans un chronoData ?

Au début du rewind, la première chose qu'on fait est de rendre kinematic l'objet joueur, c'est-à-dire qu'il ne peut pas être influencé par les forces extérieures. C'est important car nous voulons que le mouvement d'un rewind soit précis : il serait étrange de voir la boule joueur se déplacer au contact d'un objet sur la trajectoire passée.



Nous avons sur ce schéma deux **chronoData** et la position actuelle du joueur représenté par le Vector3 **basePosition** (plus précisément, cette variable est initialisée seulement au début du rewind et dans un autre cas expliqué plus tard).

L'espace entre deux **chronoData** est ce que nous appelons une **zone vide,** car c'est un espace que l'objet joueur a traversé sans enregistrer d'information : tout l'enjeu est donc de remplir ces **zones**, sans quoi, nous serions obligés de téléporter l'objet d'un **chronoData** à un autre, ce qui ne donnerait pas un mouvement fluide.

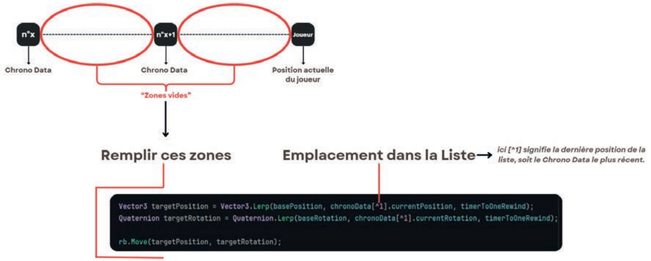
Pour remplir une **zone vide,** nous faisons une interpolation de la position de l'objet joueur entre la position de base étant **basePosition** et la position visée étant la position du dernier **chronoData** en fonction de la variable **timerToOneRewind**.

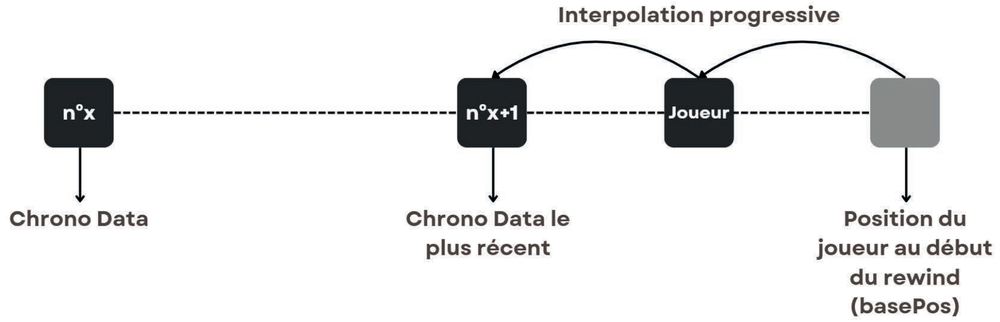
Cette variable augmente progressivement de 0 à 1 grâce à ce calcul :

```
if (timerToOneRewind < 1)
{
    timerToOneRewind += (Time.fixedDeltaTime * (rewindSpeed / distance)) * rewindAction.ReadValue<float>();
}
else
{
    timerToOneRewind = 1f;
}
```

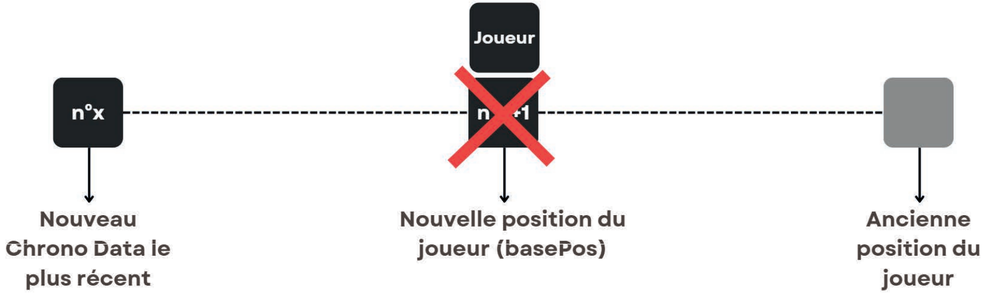
(pour l'explication du calcul, cf. Filtrage et interprétation des inputs)

Lorsqu'elle est égale à 0, la position du joueur est égale à **basePosition** ; lorsqu'elle est égale à 1 la position du joueur est égale à la position du dernier **chronoData**.





Une fois que **timerToOneRewind** est égale à 1, on détermine que le joueur est arrivé à destination. On supprime donc de la liste le **chronoData** le plus récent.



Au moment de la suppression, trois choses notables se produisent :

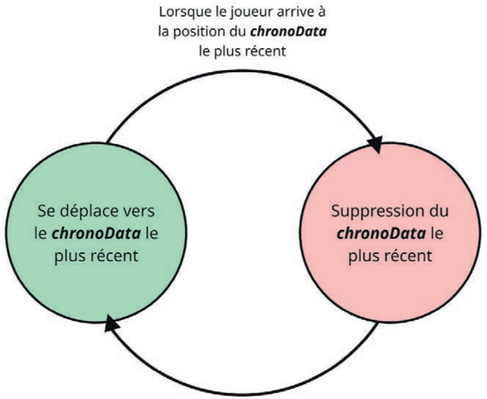
- Supprimer le **chronoData** le plus récente signifie que le **chronoData** précédent devient le plus récent du point de vue de la liste ;
- On donne la position actuelle du joueur comme nouvelle position pour la variable **basePosition** ;
- On réinitialise à **0 timerToOneRewind**.

Cela crée une boucle effet domino, où l'objet joueur se déplace vers un **chronoData**, le supprime, puis se déplace vers le prochain **chronoData** et ainsi de suite.

Il y a deux exceptions lors de la boucle :

- S'il ne reste plus qu'un seul **chronoData,** on ne peut pas la supprimer, cela a pour effet de figer le joueur à la position de ce **chronoData** le temps qu'il reste appuyé sur le rewind ;
- Si le joueur lâche l'input de rewind ou provoque un impulse, le rewind est interrompu (cf. Filtrage des inputs p.90).

Toute la procédure d'interpolation de la position de l'objet comprend en réalité aussi sa rotation.



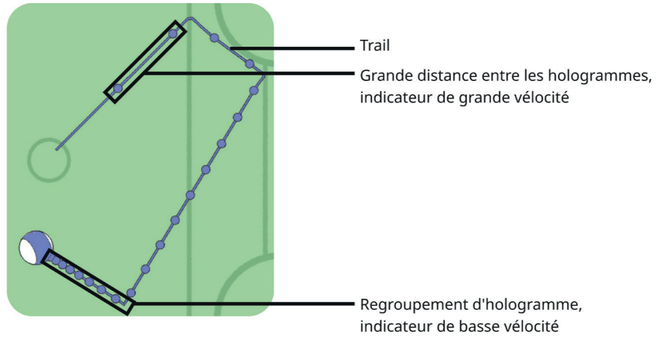
c.) La fin du rewind

A chaque itération de la boucle, le **chronoData** dernièrement supprimé est assigné au **chronoData** appelé **chronoDataToApply**, écrasant sa valeur précédente : la vélocité linéaire et angulaire y étant enregistré sont ensuite appliquées à l'objet joueur lorsqu'il termine son rewind. On désactive au préalable l'option kinematic du Ridigbody.

L'objet reprend donc sa course comme attendu : plus qu'un simple retour dans le temps spatial, c'est aussi un retour dans le temps de la vélocité et donc des forces appliquées à l'objet. Cela ouvre tout un champ des possibles, notamment sur la répétition d'une action, la synchronisation d'une trajectoire déjà terminée à un événement du présent, ou l'annulation d'une collision pour continuer sa trajectoire comme si la collision n'avait jamais eu lieu.

d.) Et les feedbacks visuels alors ?

L'idée était simple : une traînée derrière le joueur avec une apparition de sphère (que nous appelons hologrammes) toutes les x créations de **chronoData,** permettant de se rendre compte de la vélocité de l'objet en regardant l'espacement entre ces hologrammes.



Du point de vue du code, c'est ce qui a causé le plus de problèmes et de bugs : bien que le concept soit simple, en pratique cela ne l'est pas autant à cause des objectifs que nous

d.1) Les hologrammes et le système de pooling.

Sur les versions précédentes, il était parfaitement possible qu'un grand nombre d'objets soient dans la scène : or, avoir des dizaines d'objets faisant tous apparaître d'autres objets (hologrammes) en même temps est très coûteux en ressources, et donc pas du tout optimisé. On part du principe que tous ces objets bougent en même temps, ce qui n'aurait pas été le cas la majorité du temps, mais les précédentes versions étaient très différentes et c'était donc quelque chose d'envisageable.

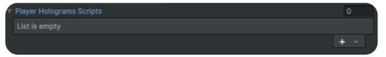
Il fallait trouver un moyen d'optimiser ce système : c'est de cette façon que nous avons créé sans nous apercevoir notre propre système de pooling.

Un système de pooling permet globalement de gérer une liste d'objets et de les activer/désactiver au lieu de les instancier/supprimer, ce qui est beaucoup moins coûteux en ressources : toute la difficulté vient surtout des conditions d'apparition des hologrammes nous permettant de déterminer quel objet de la liste activer/désactiver et téléporter à la position du joueur.

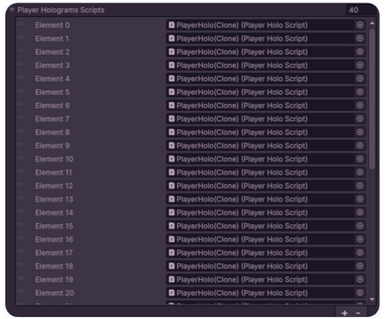
Au start, on instantie **holosToSpawn** nombre de hologrammes (ici 40) : leur script **PlayerHoloScript** est mis en cache et rangé dans une liste **playerHologramsList**, puis l'objet est désactivé.

L'avantage de ce système est que nous pouvons, en changeant le nombre maximum de **chronoData**, aussi changer le nombre maximum de hologrammes, ce qui était très pratique pour playtest et changer au fur et à mesure du projet les valeurs. C'est la raison principale qui explique pourquoi nous n'utilisons pas d'array pour ce système.

Avant de lancer le play mode

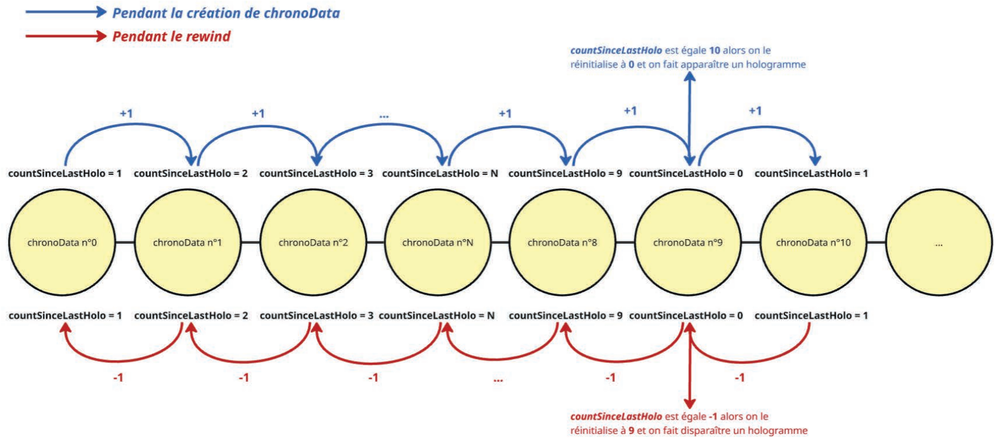


Dès la première frame du play mode



Nous avons, dès le départ, tous les hologrammes que nous allons manipuler.

Schéma : Processus interne qui détermine l'apparition ou disparition d'un hologramme



En interne, on incrémente la variable int **countSinceLastHolo** à chaque créations d'une **chronoData** : lorsqu'elle atteint la valeur **countForDataToSpawn** (ici 10), on fait apparaître un hologramme et **countSinceLastHolo** est réinitialisée à **0**.

Le joueur peut arrêter son rewind à n'importe quel **chronoData** : le compteur doit s'adapter en conséquence pour avoir la bonne valeur, sans quoi les hologrammes ne pourraient pas réapparaître aux bonnes positions. On réduit de **1 countSinceLastHolo** à chaque fois que l'on supprime une sauvegarde à cause du rewind : lorsqu'elle atteint **-1,** on la fait loop à **countForDataToSpawn - 1 (donc 9).**

Il y a deux types d'apparition d'hologrammes et deux types de disparition d'hologrammes :

- Apparition lorsqu'il y a encore des hologrammes cachés ;
- Apparition lorsque tous les hologrammes sont montrés, mais que les **chronoData** les plus anciennes sont supprimé à cause d'un surplus de **chronoData**.

On rentre dans le premier cas lorsque :

1. **activatedHolos**, qui au début est égale à **0**, est inférieure au nombre maximum de hologrammes définit dans **maxNumberOfHolos** (ici 40).
2. **countSinceLastHolo** est égale à **savesForHolo (ici 10).**

Dans ce cas, on incrémente **activatedHolos** puis on détermine la position de l'hologramme à activer dans la liste tel que :

holoScriptPos = activatedHolos - 1.

On active donc l'hologramme à cette position de la liste :

```
playerHologramsScripts[holoScriptPos].gameObject.SetActive(true);
```

De la même façon, on change sa position et rotation.

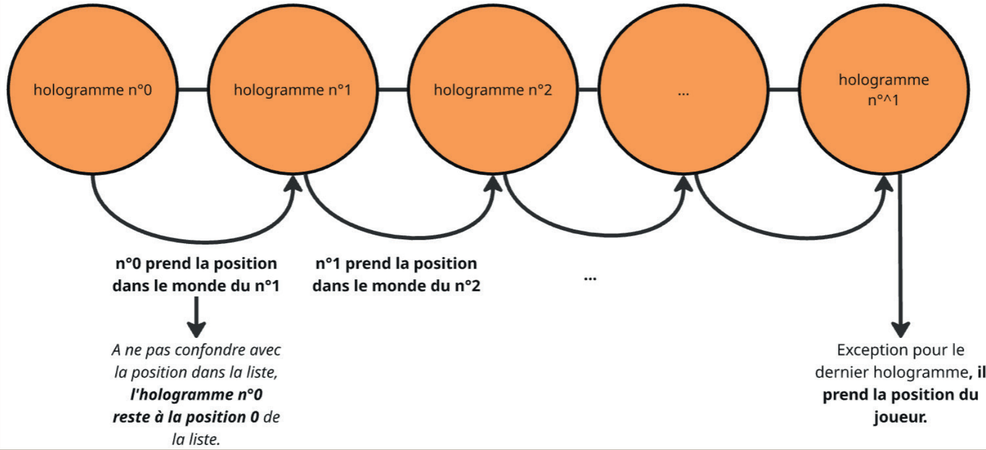
La rotation est devenue obsolète car nos hologrammes sont maintenant des sphères (cf. Sous partie : L'évolution du rewind au fur et à mesure des versions).

On rentre dans le deuxième cas lorsque :

- **activatedHolos** n'est pas strictement inférieure à **maxNumberOfHolos**, signifiant que tous les hologrammes sont activés ;
- **countSinceLastHolo** est égale à savesForHolo.

Dans ce cas, on détermine que le joueur dépasse le nombre maximum de **chronoData** : lorsque cela arrive, les données les plus anciennes sont supprimées en même temps que des nouvelles sont créées ; il nous suffit simplement de téléporter dans le monde chaque hologramme à la position du prochain hologramme respectivement dans la liste (sauf exception pour le hologramme le plus bas dans la liste (donc le plus récent), qui prend la position du joueur).

Schéma : Effet domino des positions des hologrammes




```
for (int i = 0; i < maxNumberOfHolos; i++)
{
    if (i < maxNumberOfHolos - 1)
    {
        playerHologramsScripts[i].gameObject.transform.SetPositionAndRotation(
            playerHologramsScripts[i + 1].transform.position,
            playerHologramsScripts[i + 1].transform.rotation);
    }
    else
    {
        playerHologramsScripts[i].gameObject.transform.SetPositionAndRotation(
            transform.position, transform.rotation);
    }
}
```

Il y a également deux type de disparitions :

- Disparition lorsque le joueur rewind ;
- Disparition similaire au deuxième cas d'apparition : ce cas est en réalité une apparition et disparition simultanée visuellement ; nous n'aborderons donc pas ce cas à nouveau.

On rentre dans le premier cas lorsque le joueur rewind et que **countSinceLastHolo** est réinitialisé à **savesForHolo - 1** (cf. Schéma p.86).

d.2.) La trail

La trail est un en réalité un composant line renderer sur le joueur qui est initialisé dans une for loop à chaque création/suppression de **chronoData** : elle possède autant de points qu'il y a de **chronoData,** et chaque point prend la position d'un **chronoData**.

B.) Le PlayerScript - L'impulsion et le filtrage d'input.

Autre que la mécanique principale, le **PlayerScript** reçoit aussi les inputs et les interprète pour l'utilisation du rewind et de l'impulsion.

a.) Filtrage d'input du Rewind

Lorsque le joueur appuie sur le right trigger alors qu'il y a plus de **0 chronoData**, un bool **isDoing** passe en true : lorsque c'est le cas, le rewind est en fonctionnement. Lorsque le joueur lâche le right trigger, tout un processus se lance pour que le rewind puisse s'arrêter sans accrochage.

L'avantage d'avoir **isDoing** comme garde, au lieu de simplement vérifier si le joueur appuie actuellement sur l'input, permet dans le code de timer une action qui se passe dans **Update(),** mais qui active un fonctionnement dans **FixedUpdate()**. De cette façon, il est impossible d'appuyer entre deux ticks de **FixedUpdate()** et donc d'avoir un input ignoré.

La valeur du right trigger est une valeur analogue, c'est à dire qu'à la différence d'un bouton qui ne peut être que égale à **0** ou **1**, **le right trigger peut être égale à toutes les valeurs entre 0 et 1.** On utilise cette propriété pour multiplier la vitesse de rewind en fonction de cette valeur : cela permet d'avoir une basse vitesse lorsqu'on appuie faiblement, et une haute vitesse lorsqu'on appuie fortement sur le le right trigger. Le joueur peut donc jauger son appui pour mieux utiliser la mécanique.

b.) Filtrage d'input de l'impulsion et de la direction du left stick

Le joueur a également la possibilité de se projeter dans la direction de son stick gauche :

- Lorsque le joueur appuie sur le bouton de charge alors que son stick gauche est appuyé dans une direction, une variable **holdStrength** augmente progressivement jusqu'à une valeur maximum.
- Au relâchement du bouton, l'impulsion est lancée de manière proportionnée à la valeur de **holdStrength** puis lance un **cooldown de 3 secondes avant de pouvoir être relancée.**

Le système a été pensé pour pouvoir utiliser l'impulsion à tout moment.

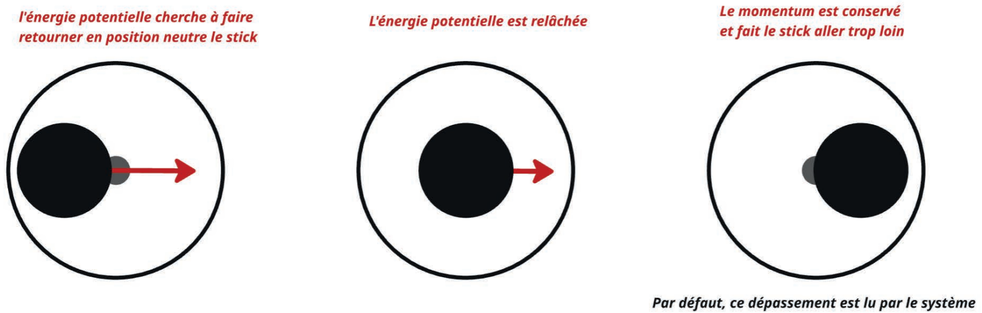
Lorsque l'impulsion est lancée pendant un rewind, le rewind est annulé de force et une variable **canRewind** passe en **false** (ce qui a pour effet d'empêcher l'activation du rewind) : le joueur est obligé de relâcher entièrement le right trigger pour que **canRewind** redevienne true.

Un f**iltrage spécial est appliqué au left stick** pour empêcher plusieurs **problèmes**.

Lors des playtest, nous avons constaté que beaucoup de joueurs avaient l'habitude de **relâcher tous les boutons et le left stick** au moment de l'impulsion.

Cela causait deux effets indésirables en fonction du timing de relâchement des inputs :

- Si le joueur relâche le stick avant de lâcher le bouton d'Impulse, le stick risque de snap back ; un snap back est un phénomène qui arrive lorsqu'on lâche un stick soudainement. Concrètement, en lâchant un stick, les composants permettent de recentrer automatiquement le stick : en revanche, l'énergie potentielle est assez puissante pour, de manière très courte, envoyer le stick gauche trop loin dans l'autre direction, ce qui est lu comme input. Cela a pour résultat d'envoyer l'impulsion dans la direction opposée souhaitée.



Par défaut, ce dépassement est lu par le système

- De la même façon, si un joueur lâche le stick avant de lâcher le bouton mais qu'il n'y a pas de snap back, l'impulse n'est simplement pas envoyé : le problème vient également du fait que lorsque cela arrive, si le joueur est en rewind, alors la vélocité appliquée à la fin du rewind est appliquée à la place (puisque beaucoup de joueurs ont l'habitude de lâcher l'input de rewind en même temps), ce qui donne l'impression que l'impulsion n'a pas été enregistré ou qu'il est partie dans la mauvaise direction.

L'enjeu est donc celui-ci : **il faut trouver un moyen pour ignorer le snapback, de plus, il faut laisser un coyote time au relâchement du stick pour qu'un très court temps après le relâchement du stick l'input de direction soit toujours lu comme activé.**

La solution trouvée résout nos deux problèmes.

La méthode **LeftStickInputFilter()** :

```
Frequently called  1 usage  BartProjectCode
private void LeftStickInputFilter()
{
    if (actualDashDir == Vector3.zero)
    {
        holdStrength = 0;
    }

    stickDir = new Vector3(leftStickAction.ReadValue<Vector2>().y, 0,
        -leftStickAction.ReadValue<Vector2>().x);

    filteredDir = Vector3.Lerp(filteredDir, stickDir.normalized, ControlSnappiness * Time.deltaTime);
    actualDashDir = filteredDir.normalized;
}
```

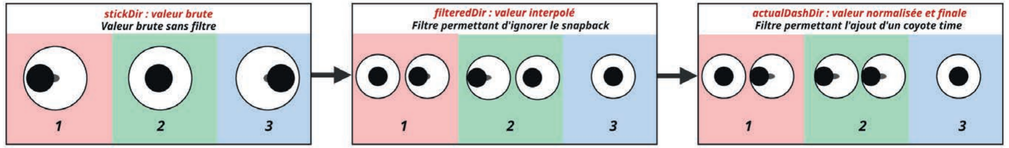
Une première valeur **Vector2 stickDir** est lue et transformée en **Vecteur3** : c'est la **valeur brute du left stick**.

La valeur **stickDir** normalisée est ensuite définie comme valeur cible de **filteredDir dans un Vector3.Lerp** : cela permet de graduellement interpoler cette valeur (vitesse d'interpolation contrôlée par la variable **ControlSnapiness**). Cela a pour effet d'ignorer les snapbacks, car **filteredDir** met trop de temps pour atteindre **stickDir,** et comme le snapback ne se produit que sur une très courte période de temps, l'input parasite n'influe pas sur **filteredDir**.

Enfin, on ajoute un filtre supplémentaire avec la valeur **actualDashDir,** qui est égale à **filteredDir** normalisée, ce qui permet d'avoir une **direction absolue instantanément au moindre changement de direction**.

De manière complémentaire, ce filtrage a aussi pour effet de créer un **coyote time** où le left stick relâché est encore lu comme activé dans une direction pendant une courte période de temps (environ 0,2s).

En 3 étapes, le snapback. Visualition de l'input interprété par le système après chaque passe de filtre.



Tous ces filtres permettent aux joueurs d'avoir une meilleure expérience sans même qu'ils ne s'en rendent compte.

C.) Le PlayerScript - Gérer la vélocité du player.

Pour éviter que le joueur ait une trop grande vélocité, celle-ci est limitée par le **Vector3 maxOverallVelocity à 20 m/s**.

Une des problématiques est que **PlayerScript()** peut créer un **chronoData** seulement si le joueur va assez vite (**minimum de 1 m/s**) : or, avant de limiter la vélocité du joueur, cela pouvait créer un bug, notamment avec les objets sphériques qui pouvaient parfois se déplacer assez lentement sans créer de **chronoData**. Nous avons donc mis en place une réduction progressive de la vitesse à partir de d'une vélocité linéaire et angulaire inférieur à **10 u/s**, et un arrêt total à **1 u/s**.

D.) Différenciation des joueurs et feedback du stick/impulsion

Passons au deuxième script : toutes les prochaines parties seront plus courtes.

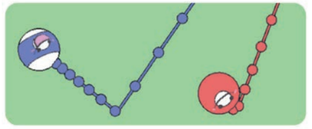
Le script **PlayerOneOrTwo** est utilisé pour déterminer l'identité du joueur vis à vis de la boule 8 (cf. page n°95). Un Enum **PlayerNumber** est définit, il contient **3 états** :

- None
- PlayerOne
- PlayerTwo

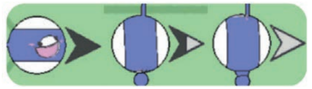
Une variable **PlayerNumber** appelée **currentPlayerNumber** est définit, et en fonction de sa valeur, le material du joueur et des feedbacks du rewind sont changés (permet la différenciation visuelle des joueurs).

Le feedback de la direction du stick est géré dans le script **LeftStickFeedback()** : elle utilise la valeur **actualDashDir** du **PlayerScript()**.

Son taux de remplissage est déterminé par la valeur **holdStrength** : sans trop rentrer dans le détail, le curseur est un fond noir avec un mask de la même forme ; on fait une interpolation de la position d'un sprite de couleur vers son offset maximum en fonction de (holdStrength / maxHoldStrength) / 2 (on divise par deux car maxHoldStrength est égale à 2), cela permet donc d'obtenir une valeur maximale de 1 (idéale pour un lerp). On obtient finalement une impression de charge.



La même stratégie est utilisée pour la représentation du cooldown :



II. La boule 8 et le système de score

La boule 8 doit entrer dans un goal pour gagner du score.

Le principe est très simple : un objet **game manager** possède le **script ScoreManager(),** qui gère tous les types de score des joueurs et leurs mises à jour visuelles.

Si la boule 8 entre dans la box collider trigger du goal du joueur P1, un point est marqué par le joueur P2.

La balle réapparaît au centre du terrain avec un offset aléatoire entre -8 et 8 sur l'axe z.

Lorsque la boule réapparaît elle devient intangible, puis se met à clignoter grâce à ce calcul :

```
alternateTimer = onGoalEnter.currentGhostTimer / (onGoalEnter.maxGhostTimer * 2);
isGhostForm = !isGhostForm;
```

Ici, **alternateTimer** est un cooldown qui réduit grâce à **Time.deltaTime** : lorsque **alternateTimer** est inférieur à 0, on lui donne une nouvelle valeur. Cette valeur est de plus en plus petite au fur et à mesure que le **currentGhostTimer** s'écoule. En même temps, on alterne le boolean qui détermine quel type de material possède la boule : cela produit un effet de clignotement de plus en plus rapide, jusqu'à la réelle fin de l'intangibilité (de la même manière que lorsque Mario prend des dégâts dans ***Super Mario Galaxy***). **isGhostForm** ici ne contrôle que le material appliqué à la boule, pas son intangibilité. L'intangibilité se termine lorsque **currentGhostTimer** est inférieur ou égale à **0**.

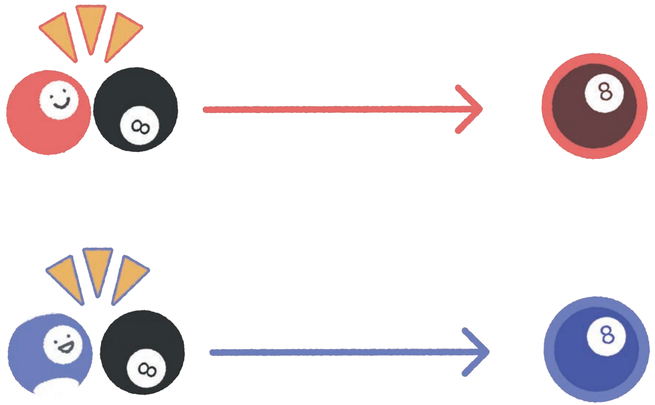
Au bout de **5 points** remporté par un joueur, celui-ci remporte une manche, les scores sont réinitialisés puis les deux joueurs sont téléportés à leur position de départ.

Après **trois manches** remportées par un joueur, la **victoire** est déclarée.

Une mécanique importante est que lorsqu'un **joueur entre en collision avec la boule 8, il en prend possession**. Cela a pour effet d'ouvrir le goal adverse et de fermer le sien.

En terme de code, la boule 8, au contact avec un joueur, détermine si l'**enum PlayerNumber currentPlayerNumber** du joueur touché est un **PlayerOne** ou un **PlayerTwo,** puis agit en fonction :

- Changement de material (Teinte rouge, bleu, grise) ;
- Activation/désactivation des bloqueurs des goals.



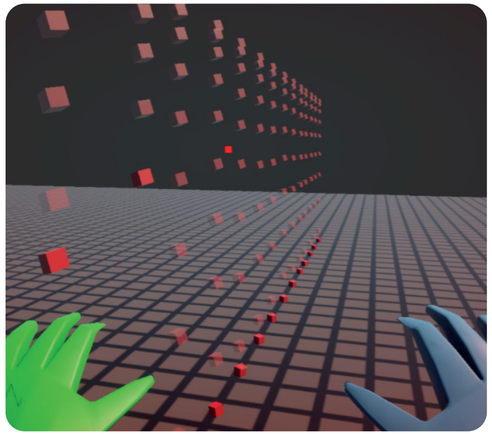
Evolution du rewind entre les versions

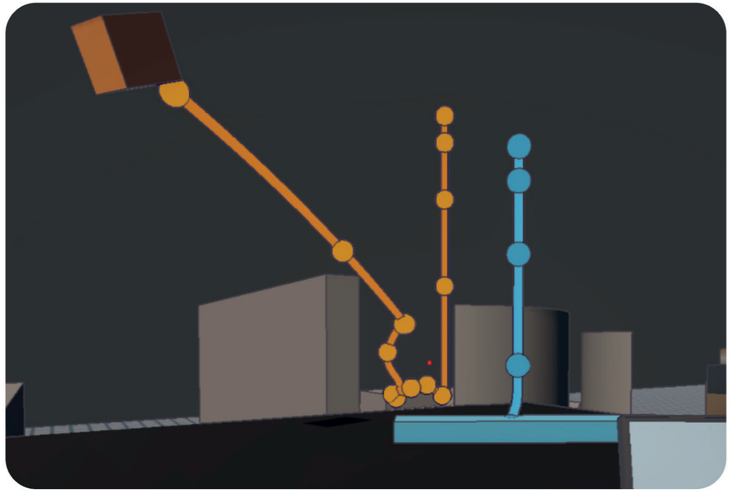
Le rewind fut notre première mécanique et son cœur mécanique a persisté à chaque version. Les grands changements se ressentent surtout dans la façon dont on avait mis en place l'interaction avec la mécanique. Nous n'allons parler que des versions où il y a eu un changement notable de la mécanique principale dans son utilisation.

V1.) FPS Parkour

La toute première idées que nous avions eue était d'utiliser la mécanique pour aider notre joueur à se déplacer sur le terrain (sauter sur des objet, se faire projeter par un objet qu'on rewind etc) : à ce stade ,on voulait s'inspirer de la chronophotographie, d'où les images rémanentes et le nom «hologramme» qui est resté jusqu'à la fin du projet.

Dans cette version, il fallait cliquer sur les images rémanentes pour faire retourner un objet dans le temps à cette cette exacte position. Les objets étaient également figés pendant environs 2 secondes avant de reprendre leur course à la fin d'un rewind pour laisser au joueur le temps de marcher dessus.





La capacité d'attraper les objets a également été introduite dans cette version, il était donc également important que le rewind fonctionne correctement pendant.

V2.) Toy en 3D

Avec cette version, nous nous sommes écartés de la chronophotographie et de l'aspect parcours.

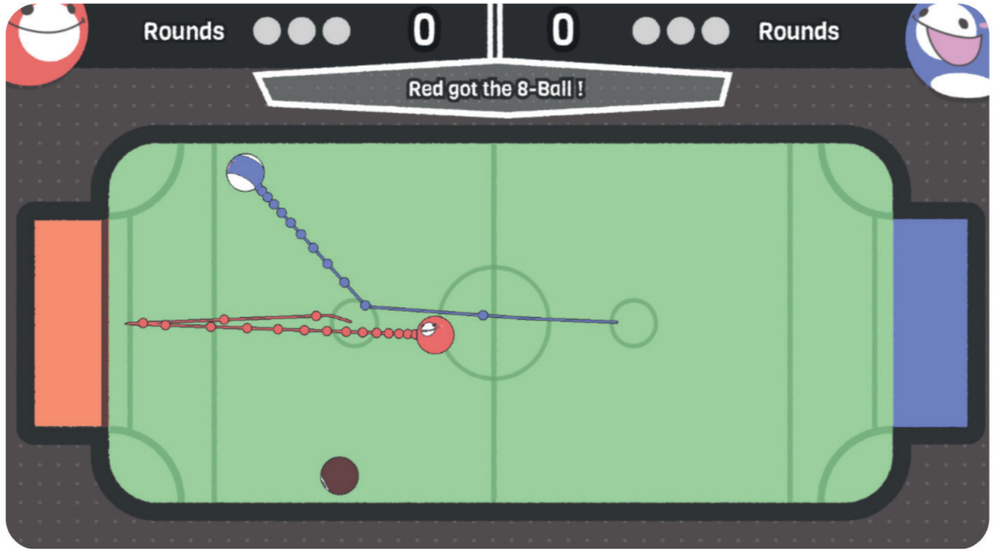
À la place de voir constamment les images rémanentes des objets, le joueur pouvait sélectionner jusqu'à deux objets maximum : un orange et un bleu.

Cette version a apporté son lot de changement, notamment la trail et le fait de pouvoir rester appuyer sur le bouton de rewind et le relâcher quand on le souhaite.

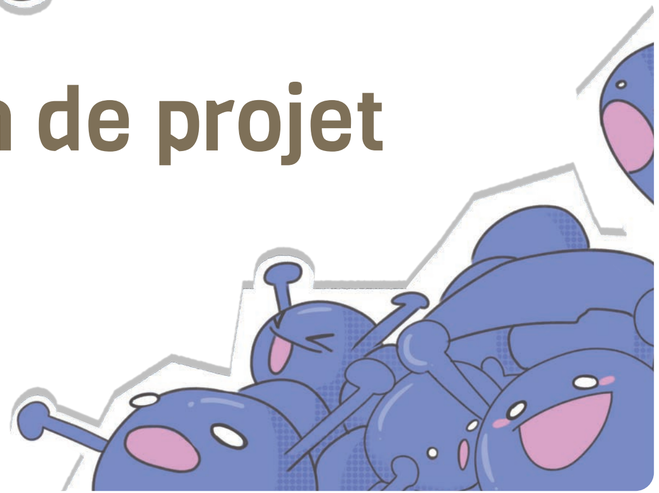
V3.) 8 o'Clock

Ceci est notre version actuelle : elle a demandé une grande refactorisation de plusieurs scripts.

Il a fallu transformer notre mécanique qui était de base pensé pour un joueur solo à une nouvelle version pour deux joueurs en multijoueur local. Nous sommes donc passé d'un joueur capable de sélectionner des objets à rewind, à un joueur capable de se rewind lui-même. Cette transformation du joueur était nécessaire pour le passage au multijoueur, notamment car nous sommes aussi passé d'un contrôle clavier souris à un contrôle manette géré par l'input manager.



Fin de projet



Pistes d'amélioration

Optimisation d'architecture

Les différentes tournures du projet ont rendu la réalisation d'une architecture globale dans les scripts très difficile, et le jeu gagnerait en clarté en ayant une réorganisation des différents éléments de code.

L'architecture globale du projet peut être grandement améliorée, notamment en convertissant certains éléments en State Machine de façon à avoir des scripts par état plus facilement modulables.

Ce travail de réorganisation n'a pu être réalisé que partiellement lors du projet, mais serait l'une des principales tâches à effectuer pour rendre le projet plus stable à des fins d'expansion pour un jeu final.

Amélioration des feedbacks et affordance

De la même manière, il y a une grande marche de progression en terme de feedbacks :

- De nouveau feedbacks apporteraient plus de diversité, notamment sur le terrain pour indiquer des effets d'Impulse et de Collision plus particuliers en fonction des situations de jeu ;
- Les feedbacks pré-existants bénéficieraient également d'un retravail, notamment afin de leur apporter un style d'animation plus abouti pour parvenir à un rendu graphique plus expressif.

De façon général, le jeu peut dans son esprit être encore plus poussé esthétiquement, mais le manque du temps nous a amené à préférer la lisibilité à la stylisation, afin d'assurer les principaux enjeux de feedbacks attendus d'un prototype.

Perspective 2v2 et Online

Enfin, en terme de Game Design, il y a une énorme marge de progression possible en fonction des axes explorés : l'idée d'un mode 2v2 semble être la première, car réutilisant la plupart des éléments de jeu déjà existans mais permettant d'approfondir les mécaniques de jeu sous un nouvel angle.

Le jeu se rapprocherait alors du babyfoot, avec la prise de rôle de chaque joueur en défense ou en attaque.

Le prototype actuel nous parraît très prometteur quant à sa possibilité à aboutir en un jeu plus complet : nous aimerions à l'avenir explorer encore plus les possibilités du jeu, et il s'agit d'un projet pour lequel nous arrivons à nous projeter dans l'avenir.

Conclusion

Ce projet a été extrêmement éprouvant, avec un reboot tardif, des difficultés à exploiter une mécanique boomerang et énormément de moments de doutes qui nous ont amené à profondément questionner notre approche avec le Game Design. Néanmoins, nous sommes fiers de son aboutissement, et notre exploitation de notre mécanique n'aurait pas été aussi plaisante sans ces moments de doute.

Il s'agit également d'un projet que nous aimerions pouvoir continuer à développer à l'avenir : après avoir vu l'engouement qu'il pouvait produire que ce soit en tant que joueur comme spectateur, nous sommes optimistes quant à la possibilité de pousser ce projet pour une sortie futur et l'application de notre vision complète pour ce projet.

Remerciements

Ce projet n'aurait pas pu voir le jour sans la contribution de beaucoup de personnes. C'est pourquoi nous voulons adresser nos remerciements :

- à l'entièreté du corps enseignant ;
- aux élèves de Game Design toutes années confondues ;
- à Esteban et Matthieu, qui nous ont aidé lors de notre phase de reboot ;
- à Luc, pour son aide et ses retours quant à l'architecture de code du projet ;
- aux playtesteurs qui ont pu nous donner leurs précieux feedbacks (et nous redonner espoir !) lors des sessions de tests ;
- à nos familles pour leur soutien, notamment pendant les moments les plus compliqués;
- à Kiara et Scarlett.