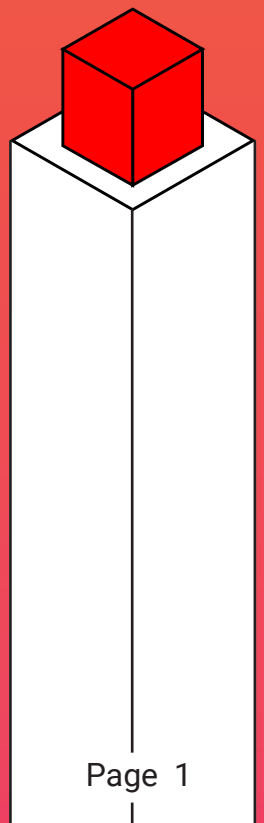
ARAKA



Quentin Gautier
André Lauzier
Thomas Vo
Stanislas Fritsch
Eden Chalon

Remerciements

Nous tenons particulièrement à remercier l'ensemble du corps enseignant de l'ICAN pour nous avoir encadré et suivi sur ce projet, nous ayant permis de produire un prototype de jeu dont nous sommes tous fier dans l'équipe.

Nous remercions également toutes les personnes qui ont accepté de collaborer avec notre groupe afin de tester notre jeu et de nous donner des retours concrets sur leurs expériences.

Nous tenons également à remercier tout particulièrement Titouan «Ønym» Drezet qui nous aura énormément aider sur la réalisation du Sound Design du jeu.

Game Design

- Intentions -
- Noyau Système -
- Références -
- Mécanique principale -
- 3Cs -
- Choix de Design important -
- RGD -
- OCRs -
- Ergonomie -

Intentions

Confronter le joueur à des puzzles basés sur du calcul et du déplacement dans un environnement ou le joueur peut directement visualiser l'ensemble du niveau.

Permettre au joueur de saisir des fenêtres d'opportunité de déplacement en fonction des blocs suivant un certain tempo.

Donner de la liberté de résolution en donnant une option au joueur de se décaler vis-à-vis du tempo pour ne pas se sentir bloqué et frustré.

Permettre d'enchainer rapidement et de façon itérative les tests de résolution du puzzle sans être trop punitif.

Noyau Système

La Tempotile: un bloc spécifique qui change d'état selon un tempo propre à sa couleur.

Tendance Système : dans Araka, le système tend à faire évoluer l'environnement en suivant le tempo des TempoTiles.

Tendance Joueur : Le joueur tend à atteindre la fin du niveau actuel.

Tension Système : La tension qui se dessine ici est celle de pouvoir finir le niveau en s'adaptant au niveau qui change constamment. La tension va fluctuer constamment entre le moment où le joueur cherche la solution et est bloqué par les changements, et le moment où il progresse dans le niveau.

Références Game Design

Références de gameplay : Déplacement

Superhot propose une méthode de déplacement unique ou le joueur définit la vitesse à laquelle s'écoule le temps en fonction de la vitesse de ses propres mouvements.
Le joueur possède alors un grand contrôle sur le tempo et le rythme de chaque niveau, et possède des phases d'analyse, des phases de planifications, et des phases d'actions, qui cyclent en boucle et changent en termes de fréquence en fonction de chaque niveau.

Pokémon Donjon Mystère exploite très bien l'idée de déplacement qui influence l'intégralité du world actuel, et qui demande au joueur de s'adapter en permanence aux agents hostiles et aux pièges qui se trouvent dans le LD.
La notion de tempo est aussi très présente, avec des ennemies qui deviennent plus fort au fur et à mesure que le joueur reste au même étage, mais également au sein de la progression du donjon entier.

Références Game Design

Références de gameplay : Terrain Evolutif

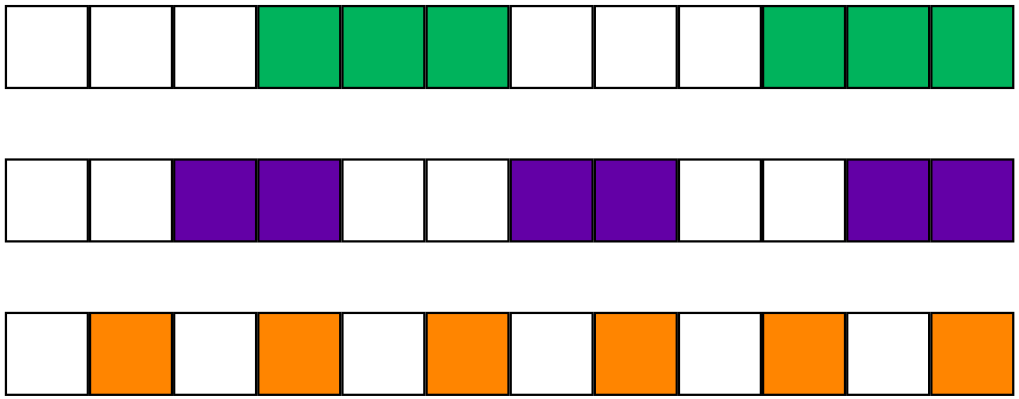
Ruined King possède une mécanique dans les combats qui modifie le fonctionnement du combat dans des «zones» dans lesquels les personnages se placent sur la timeline d'action. Ces zones peuvent être bénéfiques ou négatives, et c'est au joueur de se positionner intelligemment pour esquiver les malus ou profiter des bonus.
Cela permet de rajouter une dimension stratégique supplémentaire au jeu, sur laquelle nous pouvons nous baser pour le système d'évolution du niveau en fonction des mouvements consommés.

Baba is You propose un déplacement en grille avec des modifications variées et complexes dans le fonctionnement de chaque élément tangible du niveau.
Le joueur est limité dans ses mouvements de par le fonctionnement de son environnement et des différents éléments, mais peut à tout moment changer leurs propriétés pour progresser dans le niveau et expérimenter de nouvelles choses avec son environnement modifié.
Tout cela fait parfaitement écho à notre idée de faire en sorte que le joueur s'adapte à l'environnement qui change au fur et à mesure qu'il progresse.

Mécanique principale

Les TempoTiles

La mécanique principale d'Araka est la TempoTile. C'est une tile qui change d'état en fonction du déplacement du joueur. Ses deux états sont l'état levé et baissé. Lorsque la TempoTile est activée, elle se déplace de deux blocs de hauteur par rapport à son état baissé.



Il existe trois types de TempoTiles avec chacune leur propre couleur et leur propre rythme :

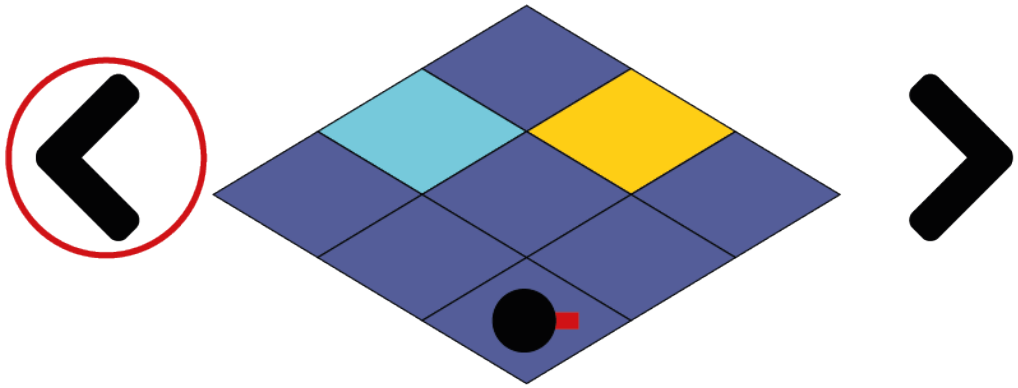
- La **Orange** change d'état à chaque mouvement du joueur.
- La **Violette** change d'état tous les deux mouvements du joueur.
- La **Verte** change d'état tout les trois mouvements du joueur.

Camera

Notre caméra est en perspective réelle, centrée sur le joueur, qui suivra donc ses déplacements. Il est possible de zoomer et dézoomer sur une certaine échelle.

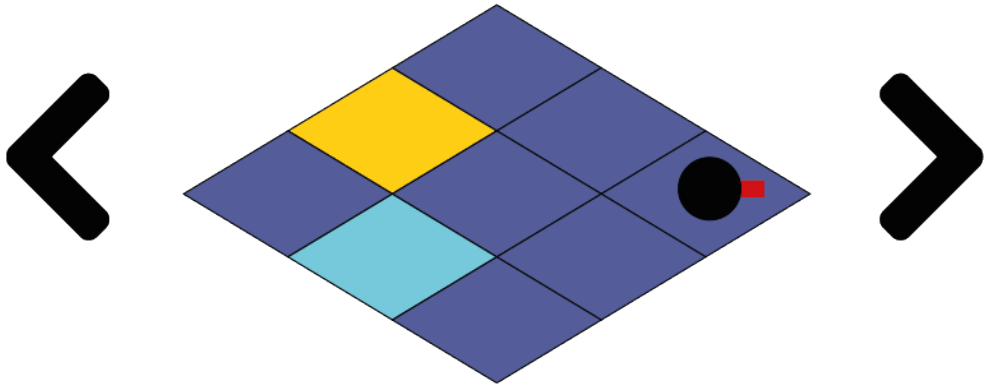
Contrôles de la caméra :
Zoom / Dézoom : écarter les doigts pour zoomer et pincer pour dézoomer.

Flèche sur les côtés : tourner l'angle de la caméra sur le niveau.



Pourquoi cette caméra ?

Nous avions fait plusieurs tests de caméra, que ce soit en degré d'angle ou en testant la perspective cavalière, qui permettait un affichage égal de toutes les tiles à l'écran. La perspective cavalière était une excellente solution pour nous, car nous avions un système de déplacement en pathfinding qui demandait au joueur de cliquer sur les tiles, ce qui rendait important le fait de les avoir toutes de la même taille pour ne pas demander un challenge trop important lié à la précision.

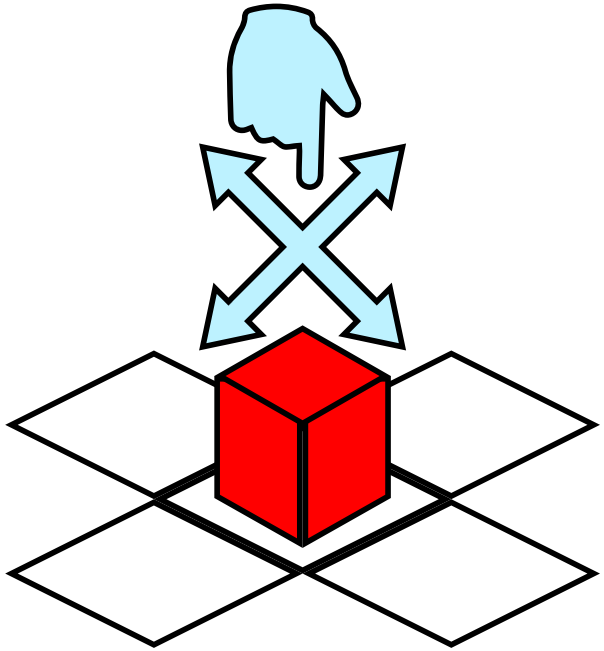


Mais avec le changement de fonctionnement de la mécanique principale, du déplacement et des niveaux,certaines dispositions de blocs rendaient les niveaux illisibles, ce qui nous a fait changer pour de la perspective réelle, car la lisibilité prime dans un jeu de puzzle.

Controls

Le joueur peut déplacer son avatar en effectuant un balayage avec son doigt sur l'écran dans une direction, le système détectant la direction enregistrée et effectuant le déplacement dans la direction la plus proche. Le joueur peut également tapoter l'écran pour se déplacer dans la dernière direction enregistrée.

Lorsque le joueur veut effectuer un déplacement impossible, vers une case inaccessible, un feedback sera envoyé au joueur pour lui indiquer que le déplacement n'est pas valide et ne sera pas comptabilisé.



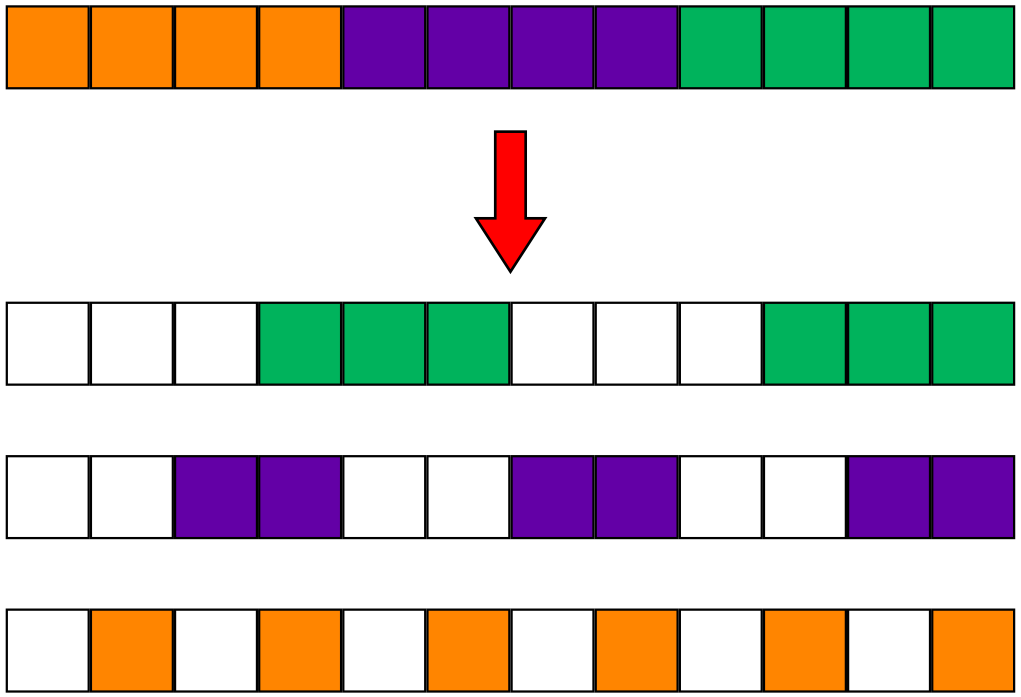
Character

Dans Araka, le joueur incarne un avatar, un petit cube rouge et noir dont le but est d'aller sur la case de fin de niveau. Cet avatar peut se déplacer case par case et interagir avec des éléments du world afin de progresser dans le niveau. Lorsque le joueur se déplace, certaines tiles du worlds se déplaceront également avec lui.

Choix de design important

Changement du fonctionnement des TempoTiles

Au début du projet, les TempoTiles fonctionnaient dans un système cyclique unique basé sur le nombre de déplacements du joueur. Les timings de changement de tempos étaient ponctuels et définis dans une boucle. Ce système marchait en tandem avec celui de retour à la position de départ du joueur lorsque celui-ci effectuait un nombre donné de déplacements. Un premier changement a été de réduire les espacements entre changements de tempo, créant des variations plus fréquentes et offrant plus de fenêtres d'opportunités au joueur.



Cependant, bien que le nombre de fenêtres d'opportunités ai augmenté, leur nature monotone n'avait que peu d'intérêt. Nous voulions de la variance dans la nature de ces fenêtres d'opportunité afin d'instaurer d'autres types d'interactions entre les différentes tiles.

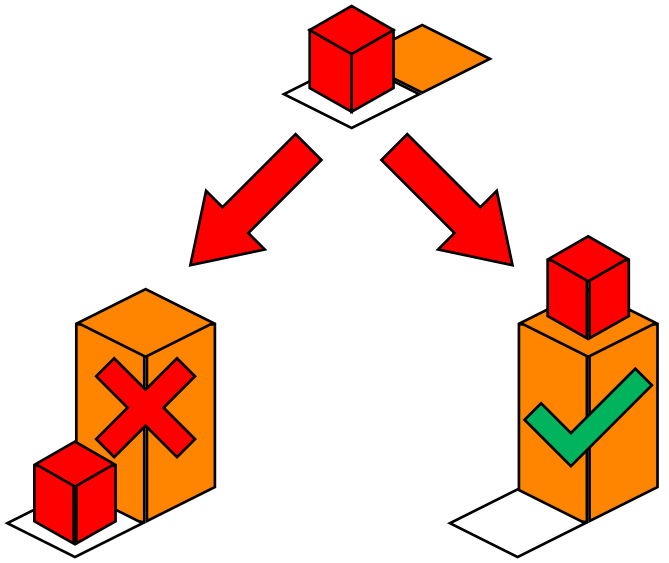
Nous avons donc choisi un fonctionnement de tiles simultanées, chaque type de tiles ayant un rythme qui lui est propre, leurs interactions créant des fenêtres d'opportunités courtes, forçant le joueur à les anticiper et le poussant à être dans le bon timing tout en créant des situations où le joueur reconnait des patterns qu'il veut éviter, renforçant l'aspect réflexion du système.

Choix de design important

Priorité Joueur - TempoTiles

Toutes les actions liées au déplacement sur le tour actuel seront effectuées après le déplacement du joueur. Cela empêche les problèmes de collision liés à une Tempotile qui s'activerait avant le déplacement du joueur.

D'un aspect purement gamme feeling, le joueur s'attend a ce que son déplacement soit la première chose qui s'effectue suite à son input, puis que l'environnement évolue suite à ce déplacement, cela enlève une grande frustration possible et lui donne le contrôle.

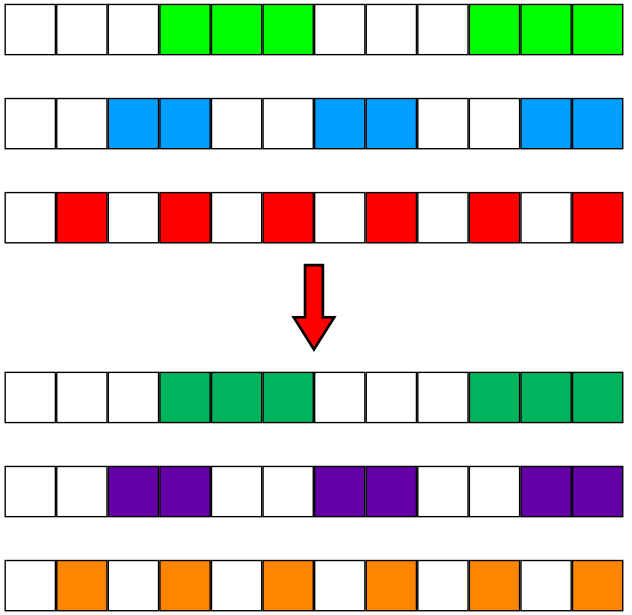


Les deux cas possibles sur lesquels on a dû en choisir un.

Couleurs des TempoTiles

Lors de l'avancement du projet, nous avons changé plusieurs fois de couleurs pour nos TempoTiles suites à de nombreux tests et retours. Les premières versions étaient beaucoup trop saturées, et comportaient des couleurs avec une mauvaise affordance vis-à-vis de ce nous voulions montrer au joueur (la couleur rouge liée au danger, les joueurs préféraient l'éviter, ou la verte qui renvoyait quelque chose de forcément positif).

Viens à tout ça s'ajouter une problématique de daltonisme, qui devait se poser rapidement, car fortement présente dans notre groupe (3 types de daltonisme différents).



Les changements de couleur effectués.

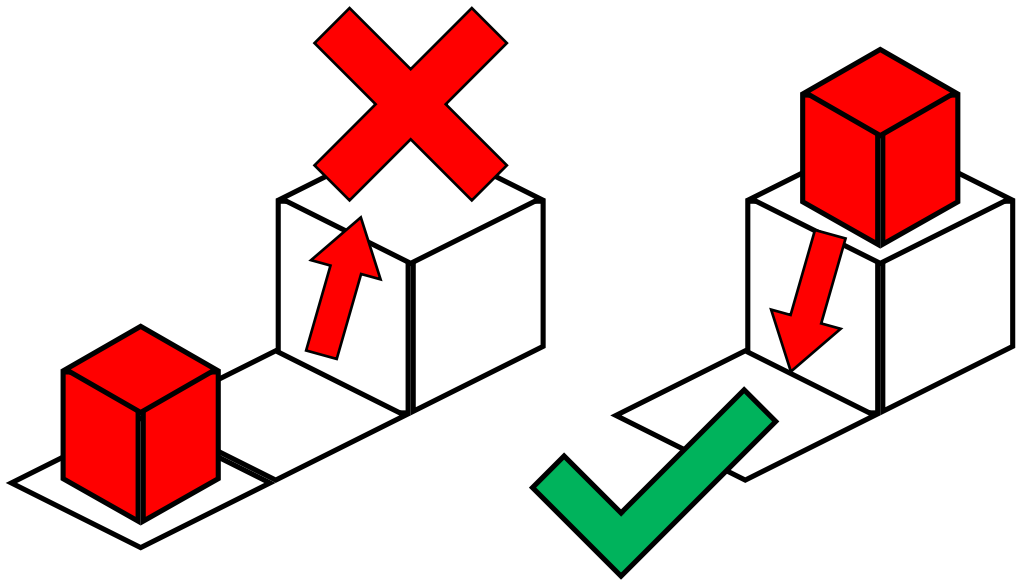
Choix de design important

Déplacements verticaux du joueur

Nous avons dû faire des choix quant à la possibilité qu'avait le joueur de se déplacer d'un ou plusieurs blocs en hauteur. Nous avons choisi de faire en sorte que le joueur puisse descendre d'un bloc de haut, mais ne puisse jamais monter en hauteur de lui-même.

Cela nous est nécessaire pour plusieurs raisons, notamment celle de créer des situations ou le joueur devra choisir ou non de descendre et donc de prendre des décisions impactantes pour la suite de son niveau.

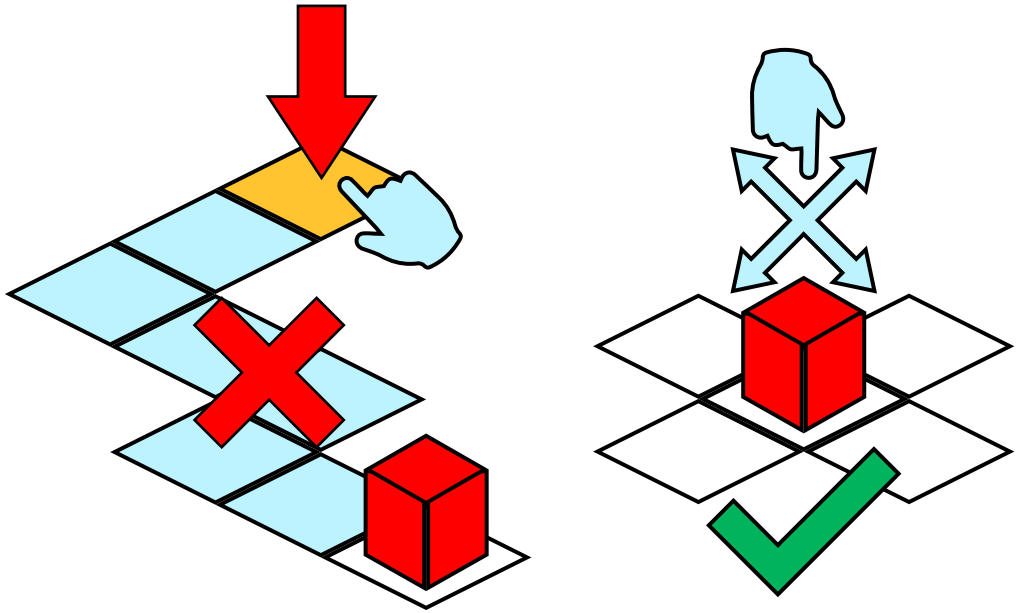
Nous avons également pris la décision que le joueur ne puisse descendre que d'un de hauteur, pour qu'il ne puisse pas descendre d'une TempoTile activée.



Choix de design important

Pathfinding ou déplacement Tile par Tile

Dans un premier temps, nous avions commencé par faire un déplacement qui fonctionnait avec du pathfinding. Cependant, au cours de la production, nous avons remarqué qu'il posait des problèmes parce que le terrain changeait dynamiquement et forçait l'arrêt du déplacement du joueur à chaque changement de tempo.



Une des compétences principales que nous voulons que le joueur mette en œuvre est le calcul et la programmation qui sont censés être employés pour optimiser sa trajectoire afin d'atteindre le plus rapidement et le plus efficacement possible la tile souhaitée. Le pathfinding est un outil trop puissant qui fait que le joueur n'a pas besoin de mobiliser ces compétences.

Le choix de passer sur mobile a également joué un rôle important sur la décision de garder ou non le pathfinding. En effet, le pathfinding nécessite des systèmes d'input différents sur mobile, changeant la manière dont le joueur interagit avec le système, la transposition des contrôles n'étant pas ergonomique.

Le pathfinding allant à l'encontre de nos intentions de design originelles et entrant en conflit avec les règles d'ergonomie sur mobile, nous avons pris la décision de le remplacer par un déplacement tile par tile.

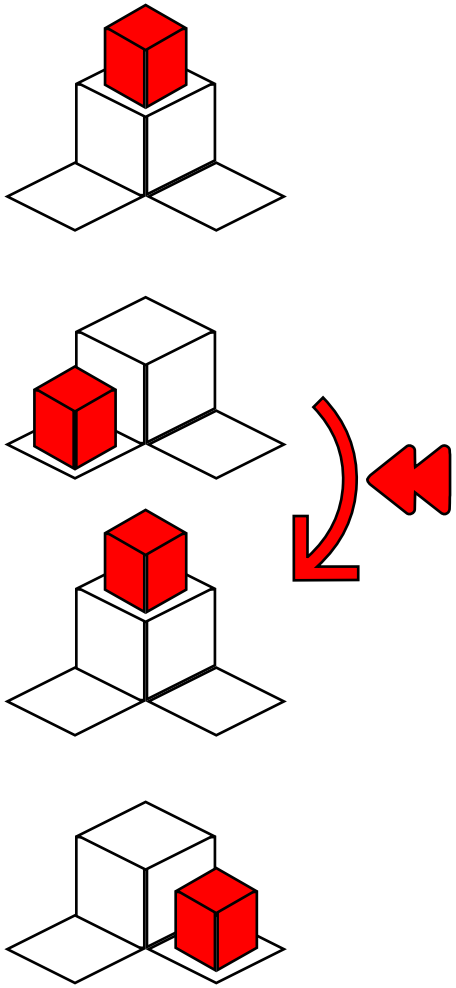
Choix de design important

Implémentation du Rewind

Lors de playtests, le cas de figure dans lequel un joueur reste bloqué sur une TempoTiles en hauteur sans aucune possibilité de descendre s'est présenté.

Notre solution face à ce problème a été de donner au joueur la possibilité de revenir en arrière. Cependant, avec cet outil, la punitivité du jeu baisse énormément, certains testeurs abusaient de la mécanique afin de finir les niveaux de manières optimales par élimination des différentes possibilités, revenant en arrière si celle qu'ils avaient choisie n'était pas la bonne.

Nous ne voulons pas que cet outil puisse être utilisé de manière abusive pour forcer les niveaux et trouver par tâtonnement sans réfléchir les solutions optimales. C'est pourquoi le rewind renvoie le joueur à sa position précédente sans impacter le nombre de déplacements total du joueur.



Signes et Feedbacks

Changement d'état des TempoTiles

Le déplacement précédent le changement d'état d'une tempotile sera précédé d'un système de particule indiquant au joueur que la tile va bouger. Ce système de particule sera différent selon si la tile va vers le haut ou vers le bas, indiqué par son orientation.

Un feedback sonore s'effectuera lorsqu'une tile proche du joueur s'abaisse ou s'élève, pour signifier au joueur qu'il y a eu une évolution et une modification.

Déplacement du joueur

Lorsque le joueur se déplace dans une direction dans laquelle il ne peut pas aller, un son s'effectue lui signalant qu'il ne peut pas aller dans cette direction, accompagné d'une petite animation du personnage et d'un court screen-shake.

Pathing d'indication pour le joueur

Pour indiquer les différents chemins possibles au sein de chaque niveau, un path sera présent au sol reliant les différents blocs que le joueur pourra parcourir.

Ce path sera dynamique pour s'adapter aux différents blocs se déplaçant (TempoTile, Ascenceur, Poussable) afin que le joueur sache constamment où il peut aller dans l'instant.

Highlight des cases disponibles

Pour compléter avec le pathing, les cases adjacentes sur lesquels le joueur peut se déplacer lors du prochain déplacement seront highlight (couleur en transparence), pour permettre au joueur de mieux se repérer dans l'espace.

RGD

Mécanique de déplacement			
But	Mécanique	Input	Challenge
Se déplacer dans le niveau et progresser vers la tile d'arrivée	le joueur choisis la direction dans laquelle le joueur se déplace sur le plan x, y	effectuer un swipe dans la direction de déplacmeent voulue	Précision : pour aller dans la direction souhaitée, le joueur doit effectuer un swipe avec un angle proche de ceui où il veut aller
			Observation : pour se déplacer de manière optimale, le joueur doit prendre en compte les éléments du niveau.
			Tactique : le joueur doit prendre en compte toutes les options de déplacement possible et choisir la meilleure

RGD

	Niveau de difficulté			
Paramètre atomique	No challenge	Facile	Difficile	Impossible
tolérance d'imprécision dans a mesure de l'angle	tolérance extrême (n'importe quel movement est reconnu)	grande tolérance (angle permissif)	petite tolérance (fine marge d'erreur)	ancune tolérance (angle absolu)
nombre de déplacement pour arriver à la fin du niveau	aucun	1 déplacement	plusieurs déplacements	une infinité de déplacements
nombre d'options possibles pour le joueur	une seule option	choix restraint d'options	blarge choix d'options	aucune option

RGD

Mécanique de TempoTiles			
But	Mécanique	Input	Challenge
Exploiter le tempo et se créer des fenêtres d'opportunités	Lorsque le joueur se déplace, les TempoTiles se déplacent sur l'axe Z en fonction du nombre de mouvements du joueur	Automatique avec le déplacement du joueur	Timing : lorsque le joueur se déplace, il doit anticiper le déplacement des TempoTiles
			Timing : chaque type de TempoTiles possède un timing de déplacement propre

RGD

	Niveau de difficulté			
Paramètre atomique	No challenge	Facile	Difficile	Impossible
nombre de mouvements nécéssaire pour faire changer une TempoTile d'état	les tiles ne changent pas d'état	les tiles changent d'état lentement	les tiles changent rapidement d'états	les tiles ne changent pas d'état (position innaccessible pour le joueur)
nombre de type de TempoTIles différents	aucun	1 seul type de TempoTIles	plusieurs types de TempoTIles	une infinité de types de TempoTiles

			Mémorisation : le joueur doit mémoriser les interactions des TempoTiles pour les utiliser de manière optimale
			Mémorisation : le joueur doit mémoriser les interactions des TempoTiles pour les utiliser de manière optimale
			Reflexion : le joueur s'appuie sur sa compréhension des situations de jeu et des patterns pour terminer le niveau
			Tactique : le joueur doit prendre en compte toutes les options de déplacement possible et choisir la meilleure
			Observation : pour se déplacer de manière optimale, le joueur doit prendre en compte les TempoTIles du niveau.

RGD

nombre de fenêtres d'opportunité différentes	une seule	nombre réduit de fenêtres d'opportunités	grand nombre de fenêtres d'opportunité différentes	une infinité de fenêtres d'opportunité différentes
durée des fenêtres d'opportunités	permanante	longue	courte	non existente
complexité des situations de jeu	une simple ligne droite	puzzle simple	puzzle complexe	puzzle impossible
nombre d'options possibles pour le joueur	une seule option	choix restraint d'options	large choix d'options	aucune option
nombre de types TempoTIles du niveau	aucune TempoTile	un seul type de TempoTile	plusieurs types de TempoTIles	Une infinité de types de TempoTiles

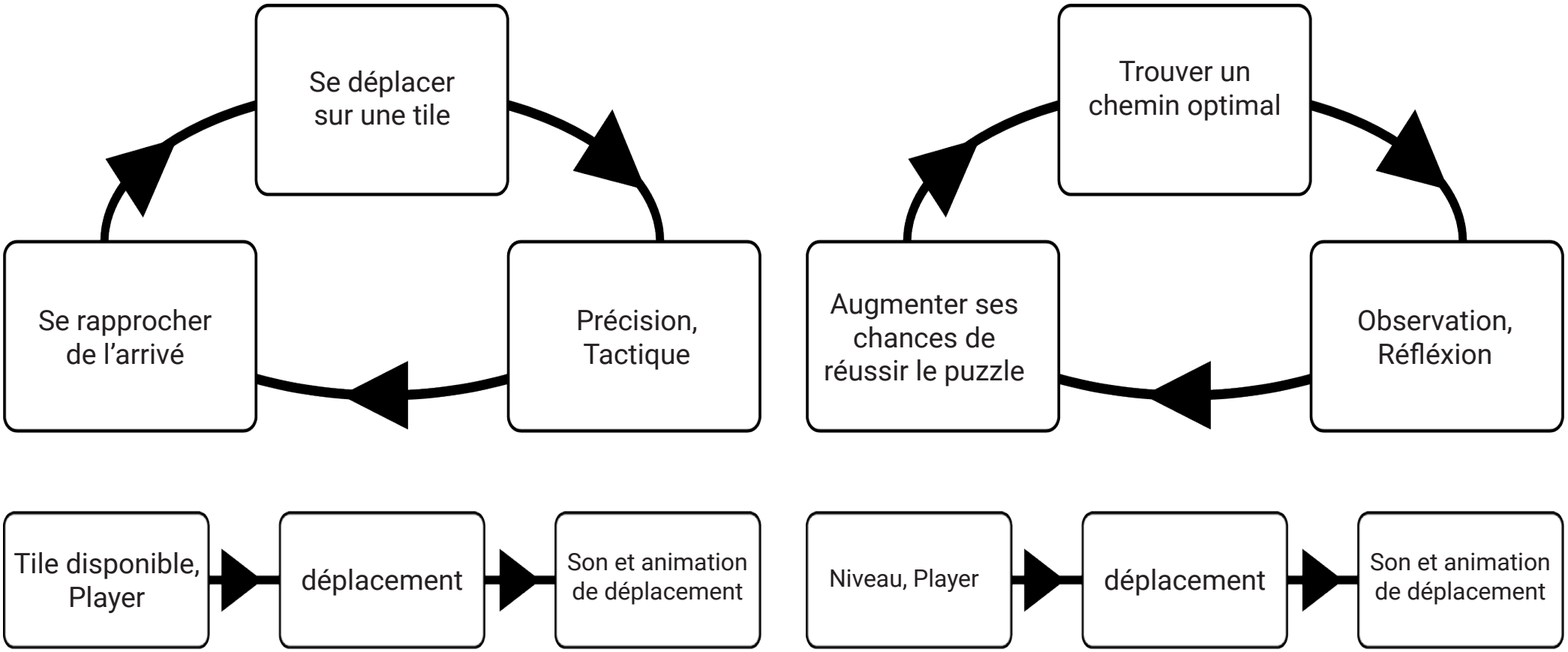
RGD

Mécanique de rotation de caméra				
But	Mécanique	Input	Challenge	Paramètre atomique
appréhender l'environnement sous un nouvel angle	le joueur peut regarder autour de l'avatar en pivotant sur l'axe Z de 90° en 90°	tapoter les boutons situés sur les cotés de l'écran	Précision : pour pouvoir faire tourner la caméra, le joueur doit tapoter sur le bouton	taille des boutons
			Observation : le joueur doit comprendre son environnement pour pouvoir utiliser la caméra à son avantage	complexité des situations de jeu
			mémorisation : le joueur utilise la caméra pour s'orienter dans son environnement	nombre d'éléments du niveau

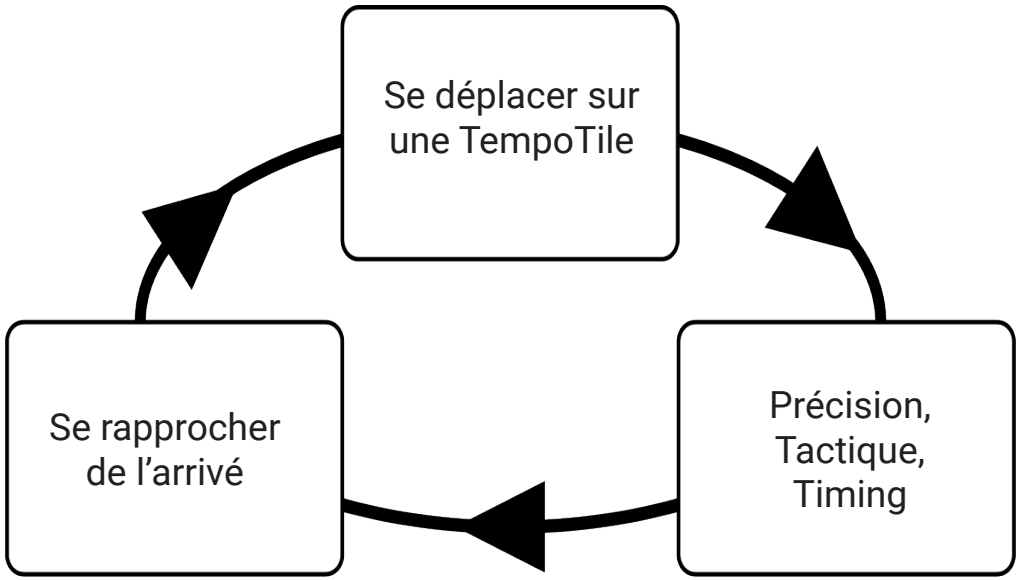
RGD

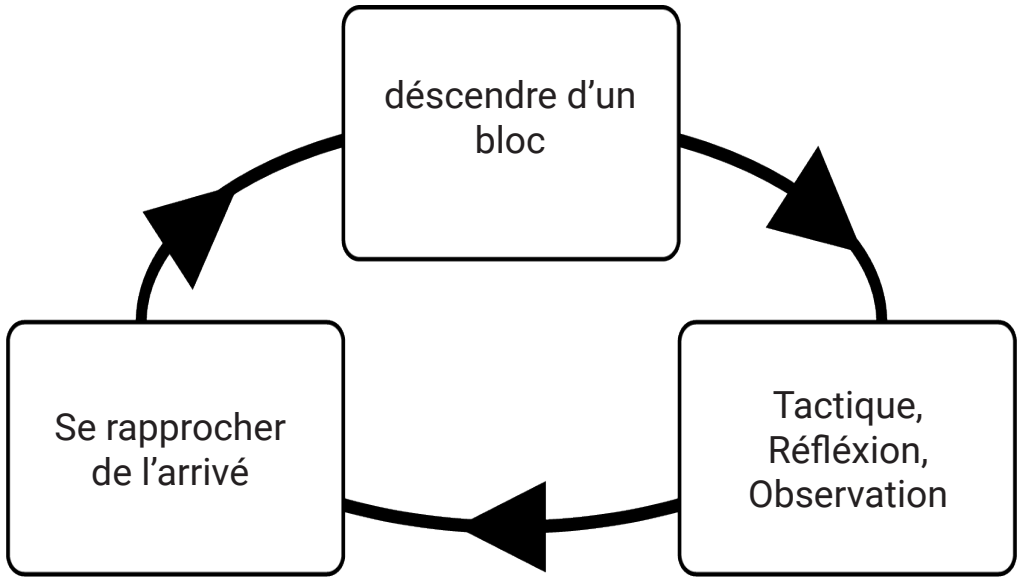
	Niveau de difficulté			
Paramètre atomique	No challenge	Facile	Difficile	Impossible
taille des boutons	tapoter n'importe où	grands boutons	petits boutons	pas de boutons
complexité des situations de jeu	simple ligne droite	puzzle simple	puzzle complexe	une infinité de fenêtres d'opportunité différentespuzzle impossible
nombre d'éléments du niveau	un seul élément	peu d'éléments	beaucoups d'éléments	une infinité d'éléments

Objectif Micro

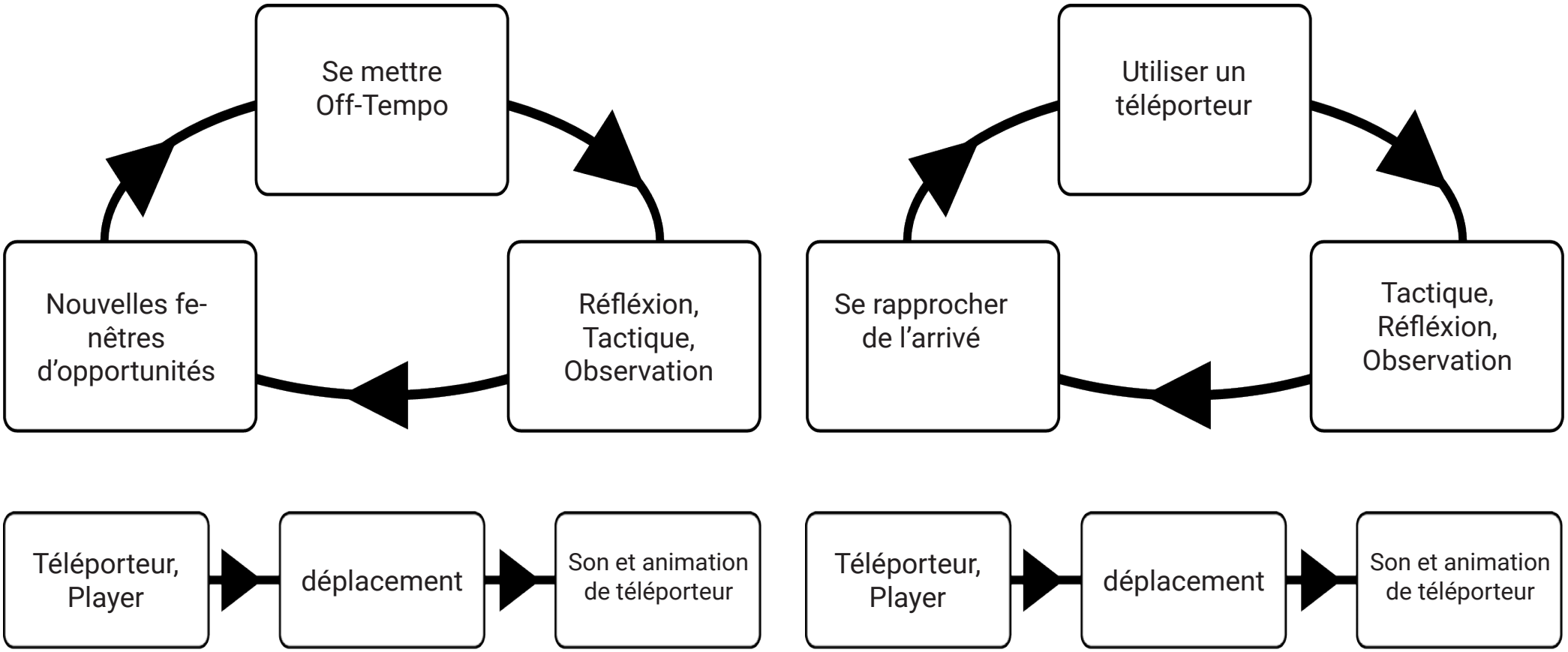


Objectif Micro

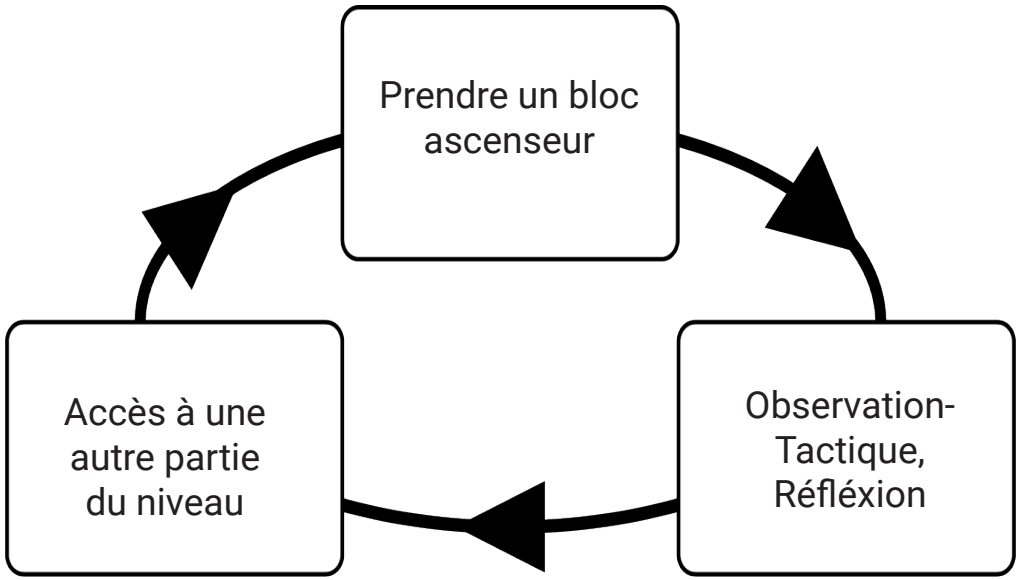


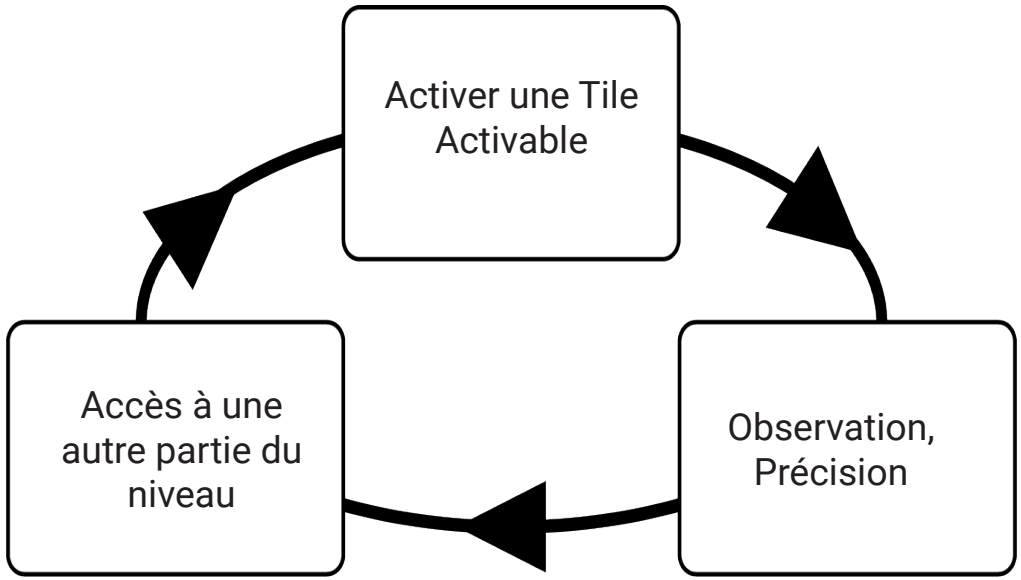


Objectif Mid

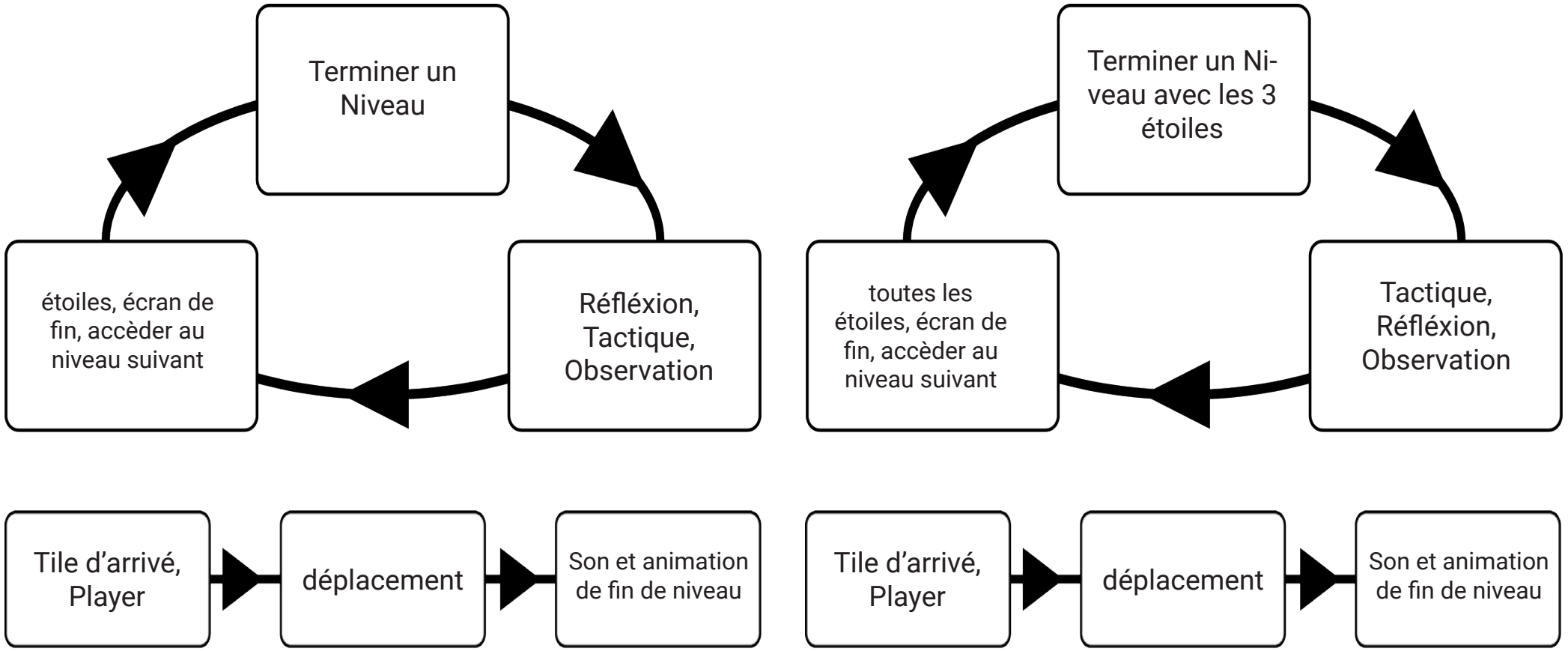


Objectif Mid

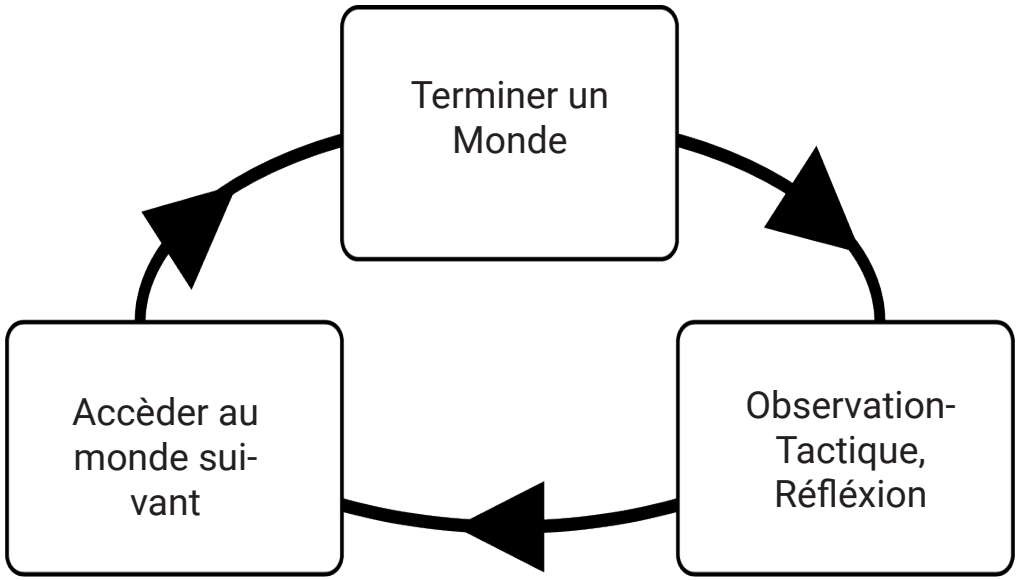


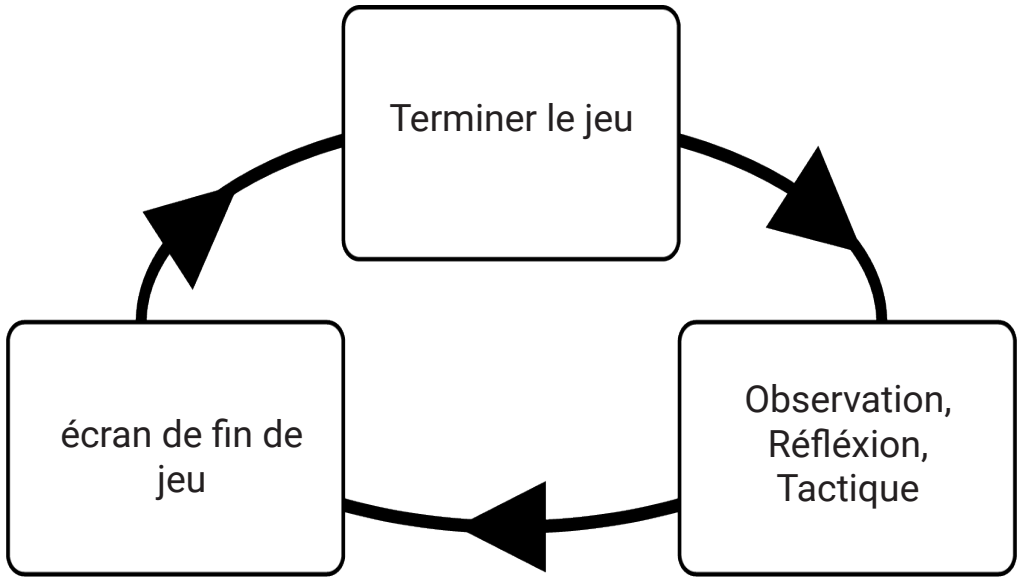


Objectif Macro



Objectif Macro





Modèle de Jesper Juul

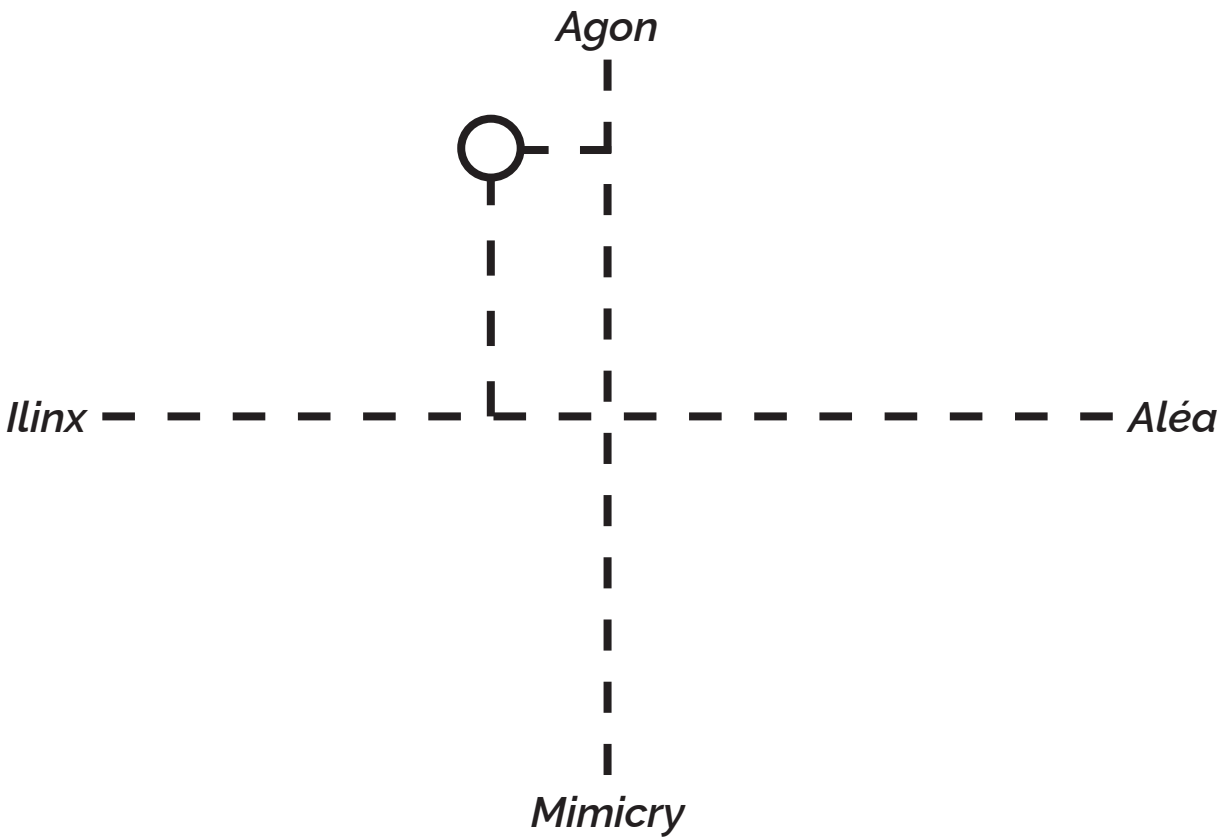
En suivant le système de classification de *Jesper Juul*, Araka est un jeu se rapprochant complètement d'un jeu progressif. En effet, Araka possède des règles bien établies de par la construction des niveaux et donc son espace de jeu réduit et sa condition de victoire.
Le jeu n'offre que très peu de liberté, l'émergence peut tout de même apparaître lorsqu'un joueur trouve un chemin plus optimisé que celui prévu par les développés.

Modèle de Caillois

D'après le modèle de Caillois, Araka se rapproche bien plus d'un système Ludus que Paidia.

En effet, le système de jeu et les compétences qui doivent être mises en jeu par le joueur sont du domaine de la réflexion, de la compréhension et de l'observation. Le jeu s'éloigne alors des compétences plus mécaniques telles que le réflexe la précision ou encore le timing, par conséquent Araka se rapproche plus d'un système Ludus d'après le modèle de Roger Caillois.

Repère de Roger Caillois :



L'aspect compétitif dans Araka se retrouve dans la complétion des niveaux pour débloquer de nouveaux mondes et donc prolonger l'expérience de jeu. Le jeu est construit sur un système d'étoile basé lui-même sur le déplacement optimal du joueur. Les étoiles obtenues permettent de débloquer de nouveaux mondes et poussent le joueur à mieux maîtriser le système de jeu.

L'ilinx peut être ressenti par rapport à l'expérience que le jeu propose aux joueurs, son univers et son système de jeu sortent de l'ordinaire et mettent en jeu des biais cognitifs utilisés au quotidien dans un contexte inhabituel.

Choix du format

Les smarphones ont deux modes d'utilisations principaux, portrait et paysage, ayant chacun leurs avantages et connotations.

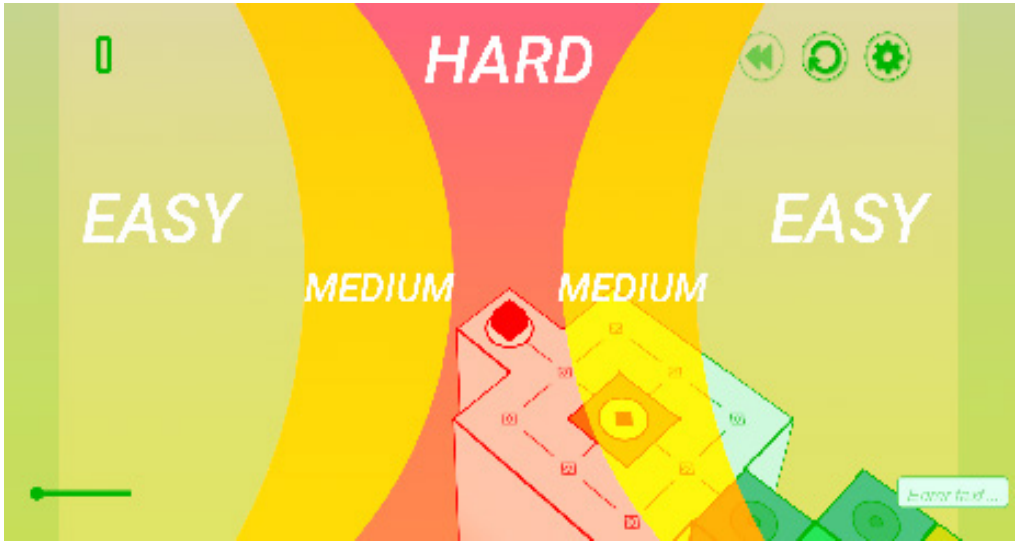
- Le format Portrait permet l'utilisation à une main du smartphone et favorise l'affichage des informations, la lecture de texte et la hiérachisation verticale. Il est souvent associé aux jeux casuals et aux applications basées sur le texte comme les réseaux sociaux, ou les applications de recherches. La position de jeu verticale peut être instable pour des séssions longues.

- Le format Paysage permet une utilisation à deux mains du smartphone et favorise l'affichage des images, des vidéos et la hiérarchisation horizontale. Il est souvent associé aux lecteurs de vidéos et d'images ou aux jeux mid et hardcore. La position de jeu à deux mains est plus propice aux longues séssions de jeu.

Compte tenu de la nature et de la cible d'Araka, nous avons choisis un format paysage favorisant la lecture de l'image et le confort du jeu à deux mains.

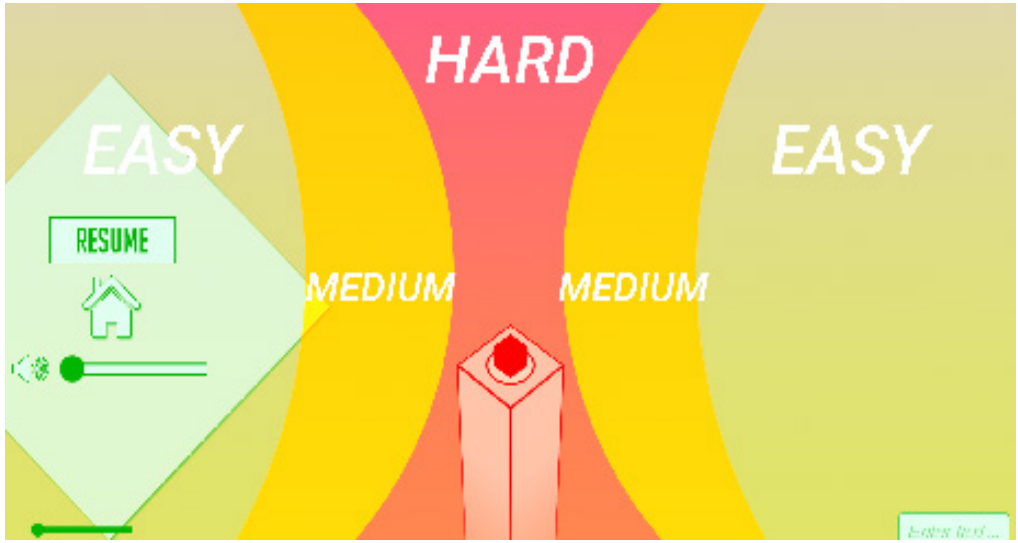
Confort de jeu - In game

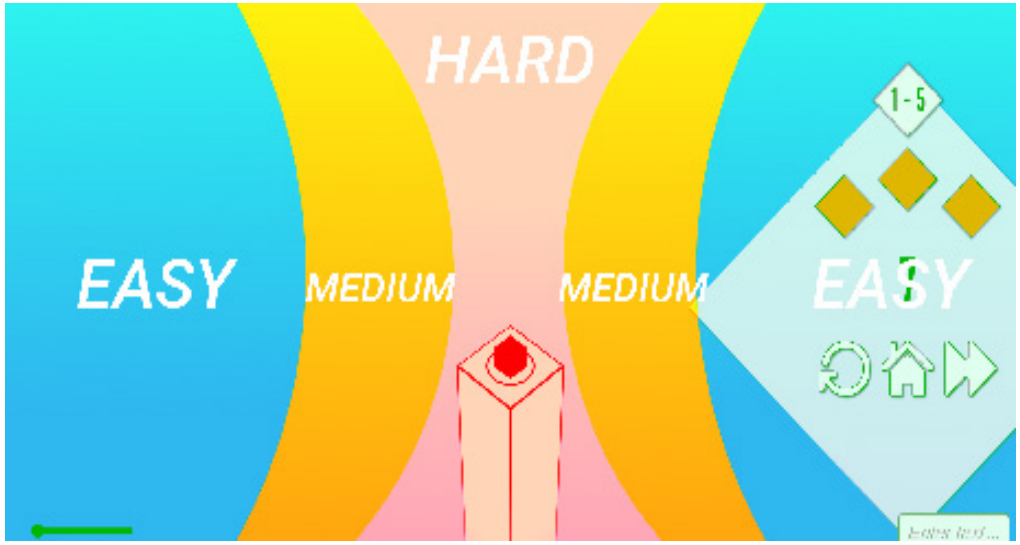
En jeu, le joueur peut swipe n'importe où sur l'écran afin de déplacer son avatar. Les deux boutons de rotation de la caméra sont situés sur les cotés de l'écran, endroit le plus facile d'accès pour le joueur afin de l'encourager à s'en servir. Les boutons pour le rewind et recommencer le niveau son à porté de sa main droite tandis que le menu pause est facilement accessible par la main gauche. Aucun bouton n'est au centre haut et bas de l'écran, favorisant le confrot du joueur au maximum.



Confort de jeu - Menus

Les menus d'Araka suivent aussi cette construction. Le menu pause et tous ses boutons sont positionnés de manière à être facile d'accès avec le pouce gauche du joueur tandis que l'écran de fin de niveau est positionné de sorte que tous les boutons soient cliquables sans efforts par le pouce droit du joueur.





Level Design

- Intention -
- Références -
- Bloc LD -
- Construction du LD -
- Signalétique -
- RLD -

Intentions

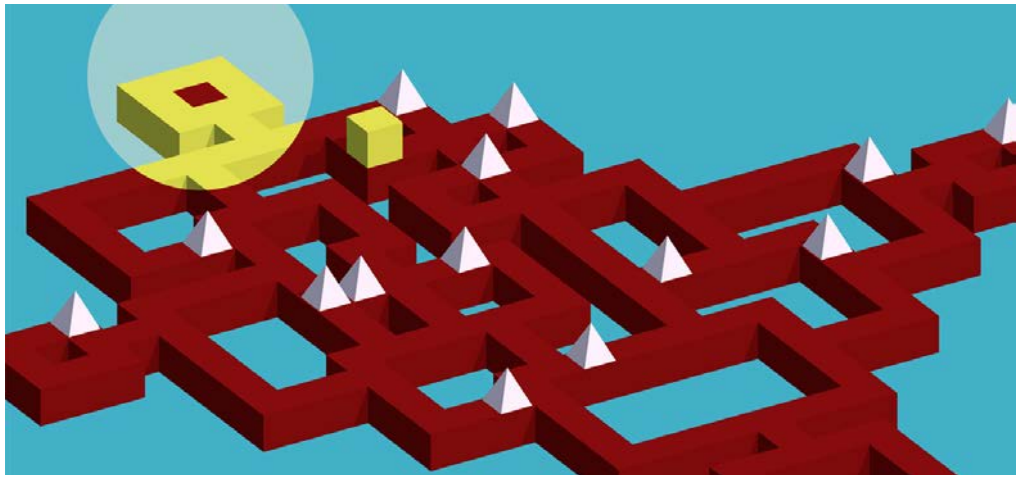
Confronter le joueur à des puzzles basés sur du calcul et du déplacement dans un environnement ou le joueur peut directement visualiser l'ensemble du niveau.

Permettre au joueur de saisir des fenêtres d'opportunité de déplacement en fonction des blocs suivant un certain tempo.

Donner de la liberté de résolution en donnant une option au joueur de se décaler vis-à-vis du tempo pour ne pas se sentir bloqué et frustré.

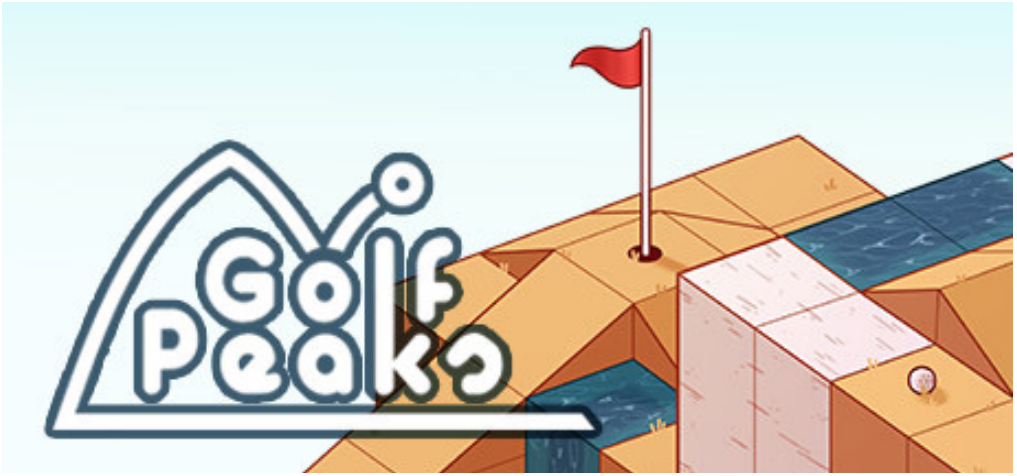
Permettre d'enchainer rapidement et de façon itérative les tests de résolution du puzzle sans être trop punitif.

Références



Vectronom est un jeu de rythme ou le niveau évolue en suivant le tempo de la musique, créant parfois des vides, déplaçant des agents hostiles, etc. La construction du LD suivant un tempo précise est très intéressante pour nous, car les TempoTiles fonctionnent sur la même base, ce qui va nous permettre de nous inspirer des patterns de constructions et des flows d'évolutions présents dans le jeu.

Golf Peaks met en avant la programmation d'action et l'observation du LD tout en ayant une quantité de déplacement et d'action limités.
Les déplacements proposés sous la forme de cartes ne permettent au joueur que très peu de solutions possibles, ce qui apporte un véritable système de «die & retry» durant lequel le joueur va tester toutes les possibilités qui lui sont proposées.
Les différents éléments de gamme et level design ne sont jamais expliqués au joueur, c'est à lui de tester et de voir les réactions et les fonctionnements pour pouvoir ensuite revenir en arrière et tenter de résoudre le puzzle avec ses nouvelles informations.



Références

Lara Croft GO reprend notre idée de pathing sur une grille avec des puzzles basés sur le déplacement, en proposant une belle courbe de progression et d'apprentissage.
Le principe de bloc ennemi est aussi une source d'inspiration pour nous, appuyant encore sur le principe de déplacement et de positionnement. Les blocs à déplacer grâce à des leviers nous ont également beaucoup aidés sur le design des TempoTiles, des Ascenseurs et des Boutons.

Hitman GO est dans la même lignée que son antécédent Lara Croft, avec beaucoup de challenge basé sur le déplacement.
Les éléments sont cependant différents, avec par exemple les trappes utilisées pour se décaler dans le timing du niveau, qui nous a inspiré nos blocs Téléporteur.

Blocs LD

Le bloc Téléporteur

Le téléporteur permet de se déplacer instantanément d'un point A vers un point B en ne consommant qu'un seul déplacement. Cela permet deux choses :

- Décaler le rythme en utilisant un déplacement compté comme unique pour se déplacer de plusieurs tiles.

- Effectuer un déplacement pour se situer dans un autre endroit du niveau, parfois inatteignable sans son utilisation.

Le bloc activable

Le bloc activable permet de faire apparaitre et/ou disparaitre certains éléments de LD de tout genre (téléporteur, TempoTiles, tiles classiques, etc.) lorsque le joueur passe dessus. Une fois que le joueur quitte la le bloc, tout ce qu'il a fait apparaitre et/ou disparaitre reste en l'état. Si le joueur repasse sur le bloc, l'effet est inversé. Cela permet de :

- Créer des niveaux beaucoup plus lisibles en séquençant l'apparition du niveau.

- Demander au joueur une nouvelle méthode de réflexion sur son pathing.

- Diviser certains niveaux en plusieurs parties.

Le bloc Ascenseur

L'ascenseur est un bloc qui s'élève et s'abaisse (selon son état actuel) de 2 blocs de hauteur lorsque le joueur le quitte.
Cela permet de :

- Créer des ponts à usage unique pour le joueur.

- Créer des chemins alternatifs d'exploration et de résolution de puzzle (ex. : escalier).

Blocs LD - Envisageable

Le bloc Poussable

Le bloc poussable est un bloc que le joueur peut déplacer en émettant un input de déplacement dans le même sens que le bloc poussable.
Il sera déplacé s'il n'est pas gêné par un mur, et peut se pousser dans le vide pour créer une plateforme. Il peut également se placer sous un bloc Tempo pour bloquer son chemin. Cela permet de :

- Proposer un challenge supplémentaire basé sur le déplacement.

- Créer des changements de positions des blocs Tempo et des blocs Ascenseur.

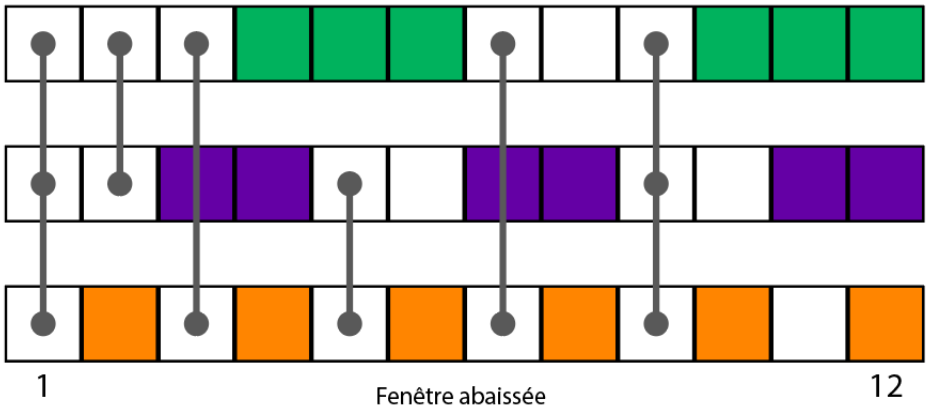
Le bloc hostile

Le bloc hostile est un bloc pouvant détruire le joueur et le ramener au début du niveau si le joueur est face à lui, à la même hauteur.
Le joueur peut également le détruire en rentrant en collision avec lui dans un autre angle que celui de face.
Cela permet de :

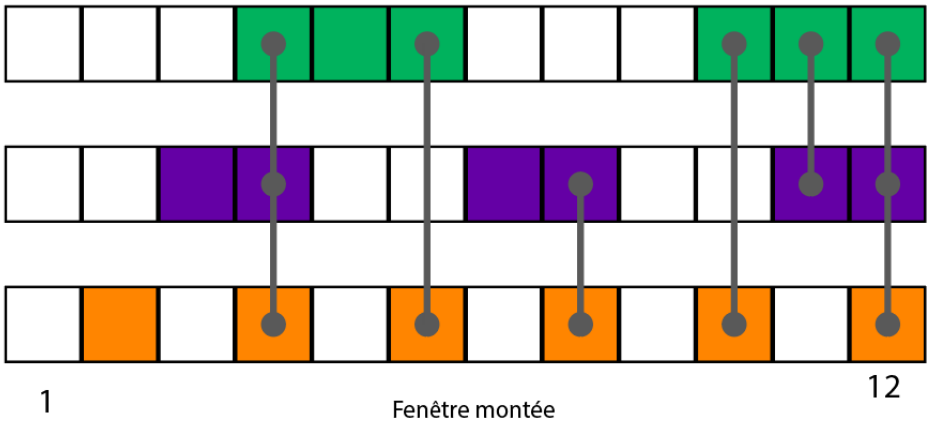
- Créer une véritable tension sur les déplacements du joueur.

- Bloquer temporairement des chemins jusqu'à ce que le joueur détruise l'hostile.

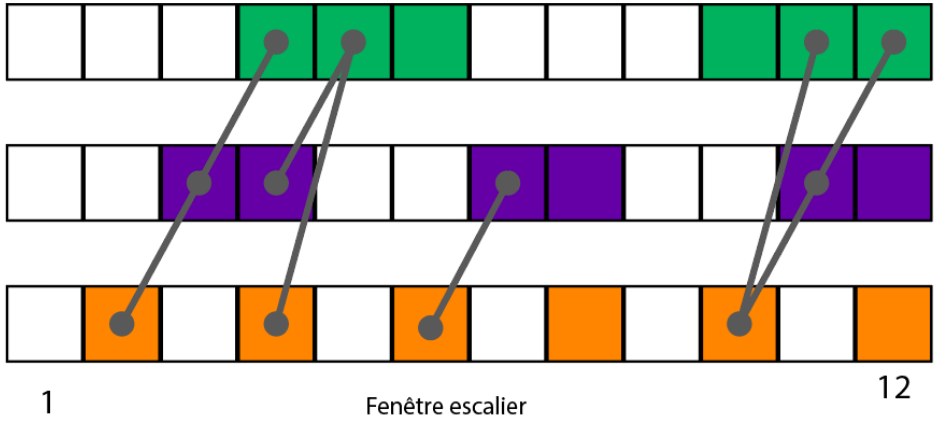
Construction en fonction des TempoTiles



Fenêtre abaissée



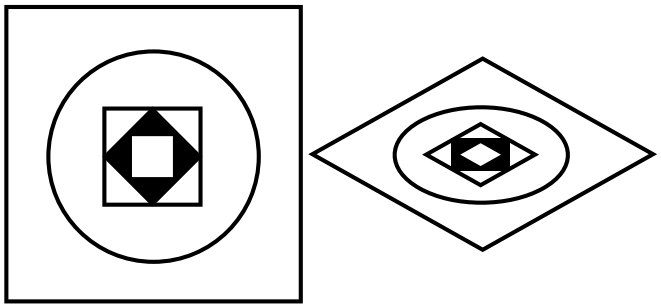
Fenêtre montée



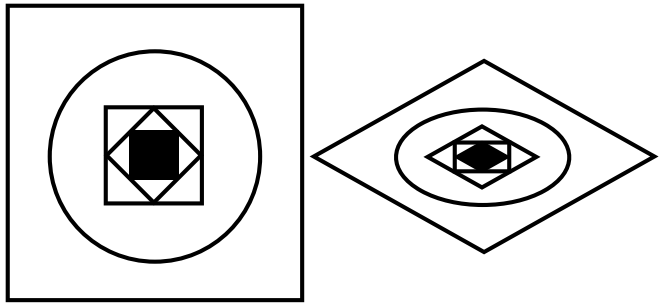
Fenêtre escalier

Signalétique

Le bloc Arrivée



Le bloc Départ

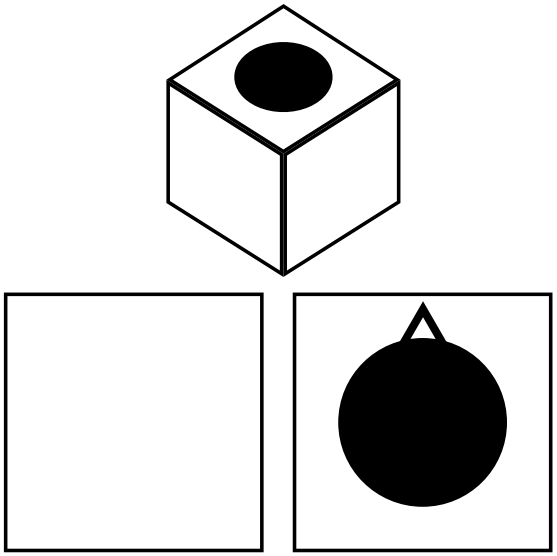


- Même type de motif que les tiles normales, en plus détaillées, pour signifier au joueur qu'il atteint la fin du chemin dessiné par le niveau.

- Système signalétique complémentaire d'emboitement pour donner une signification supplémentaire au joueur.

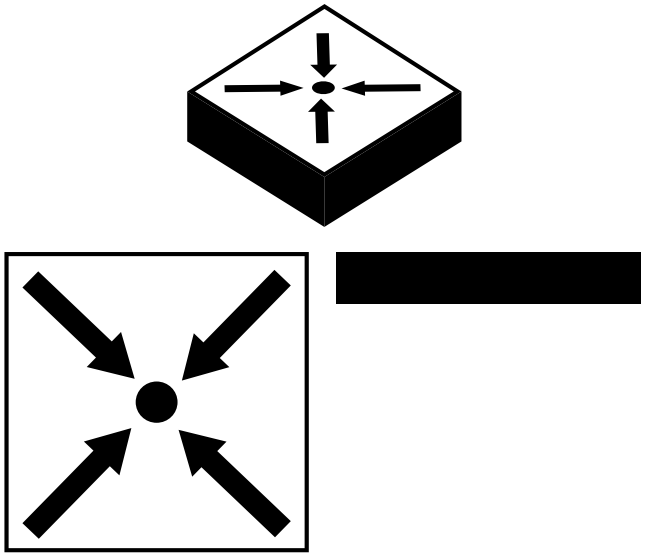
Signalétique

Le bloc Téléportation



- Fonctionnement de type tuyau : faire comprendre au joueur qu'il va s'enfoncer dedans pour ressortir ailleurs -> vide noir.

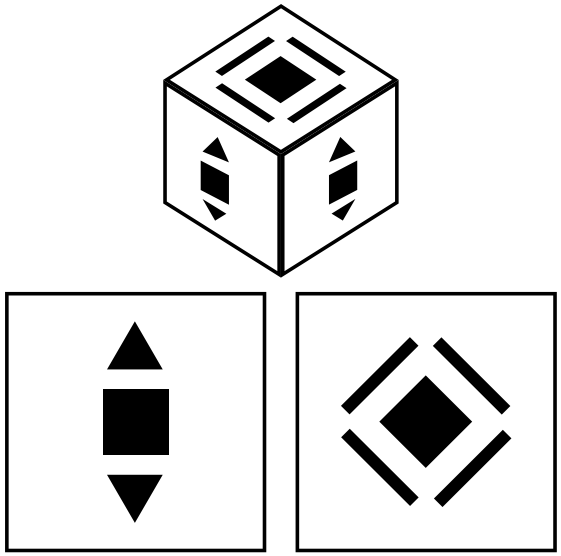
Le bloc activable



- Léger relief par-dessus une tile pour faire comprendre le fonctionnement d'appuyer dessus.

- Motif semblable à une cible pour attirer l'oeil du joueur et lui faire comprendre que l'objectif est d'arriver dessus.

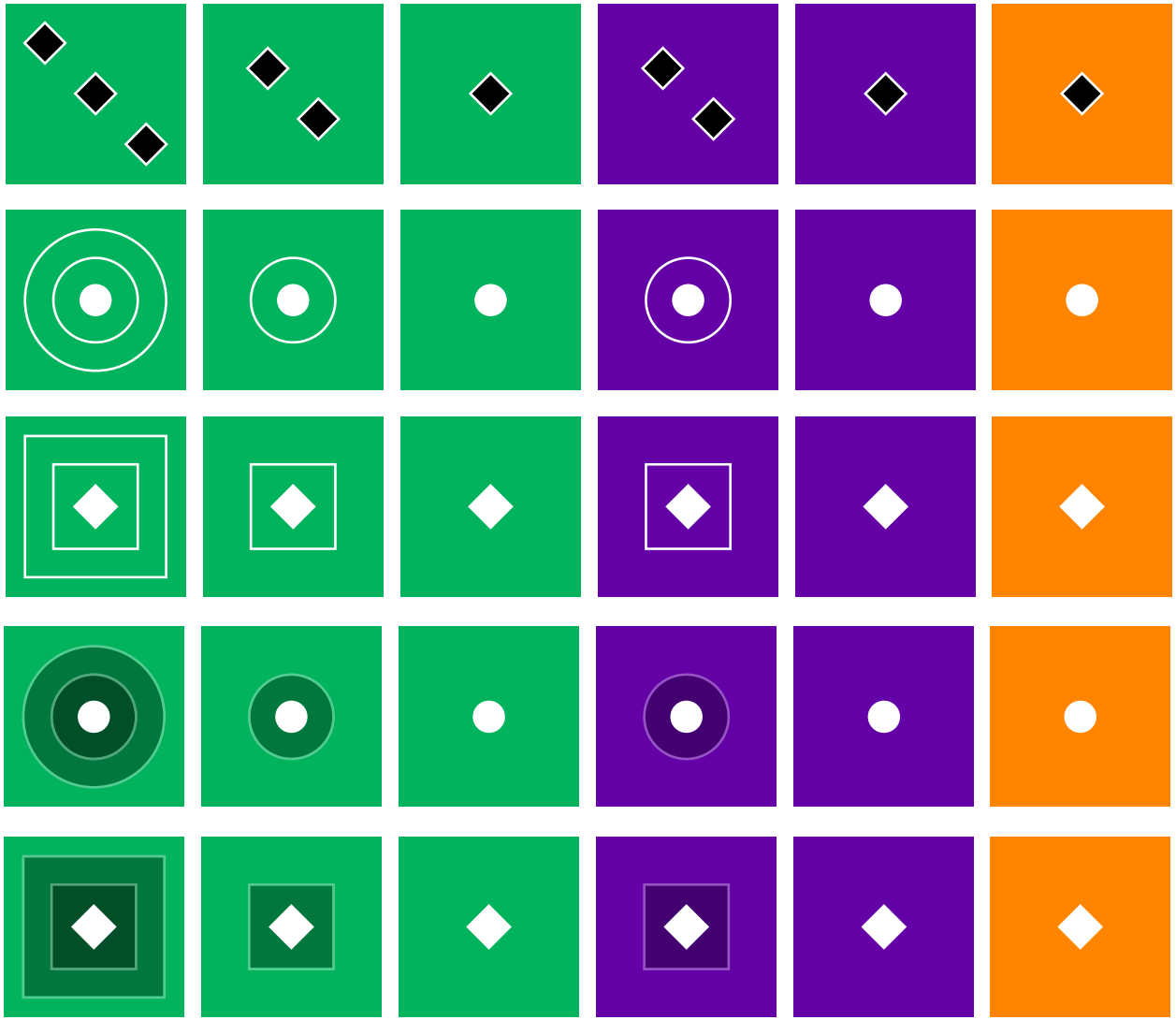
Le bloc Ascenseur



--Indiquer au joueur que c'est un nouveau type de tile. Lui faire comprendre que la tile monte et descend par son passage.

Signalétique

Les TempoTiles



Signalétique

Les TempoTiles - piste sélectionnée

- Dégradé de noir transparent pour donner de l'importance à la décrémentation.

- Cercle pour casser avec les signes principaux en carré, ce qui donne plus d'importance, d'attache à l'oeil et de lisibilité.

- Problématique de la TT orange : puisque la TT orange bouge à chaque déplacement, un simple point blanc dessus fait rapidement comprendre au joueur son fonctionnement, bien qu'il n'y ait pas de changement d'état.

Tableau

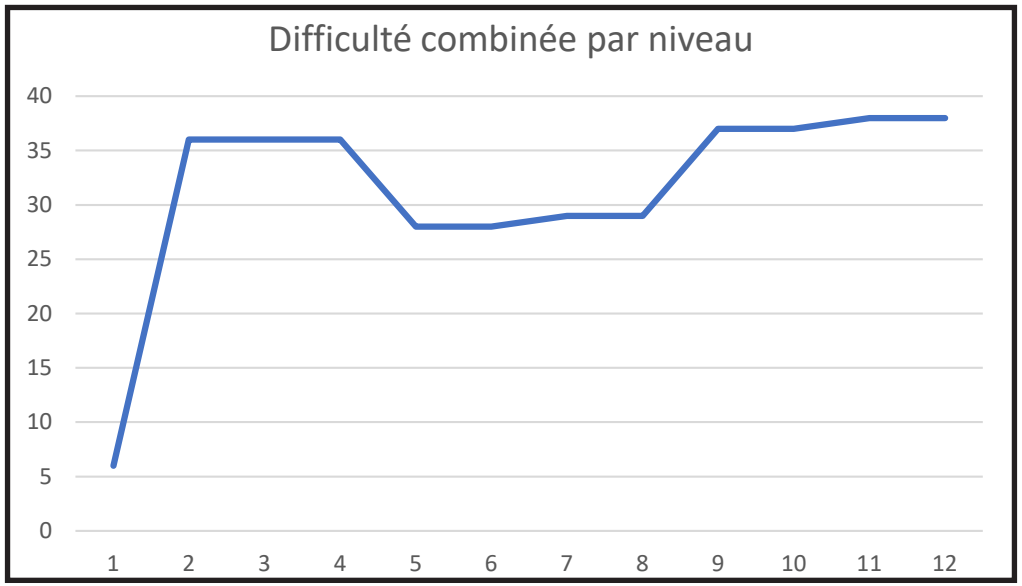
Voici le tableau affichant toutes les difficultés en fonction des situations présentent dans les niveaux du World 2 de Araka, chaque difficultés est traité du point de vue de la progression dans le niveau par le joueur et donc prend en compte le cheminement du joueur. Les difficultés ne sont pas analysé de manière extraite de leur contexte.

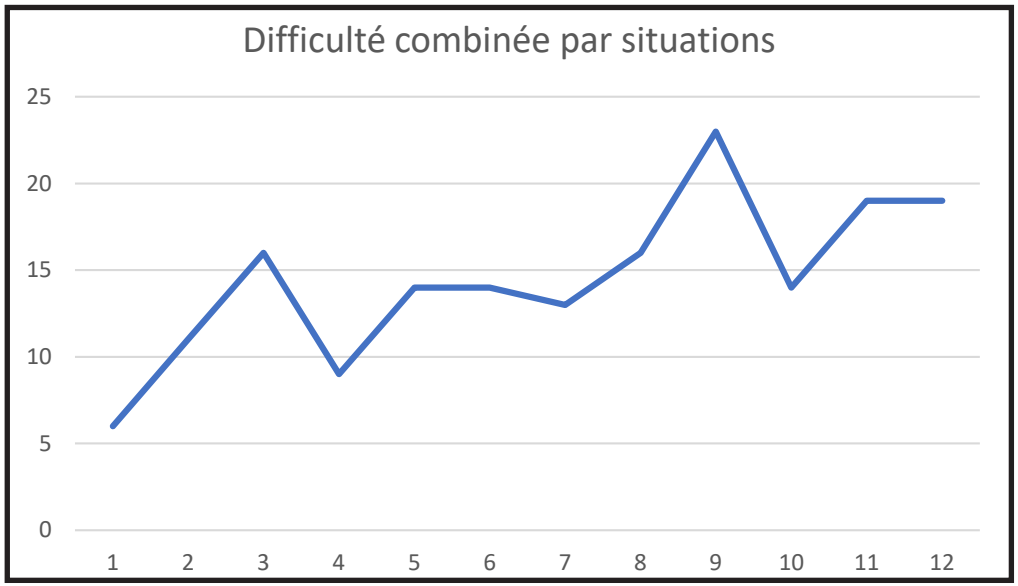
Un très faible nombre d'opportunité réduit drastiquement la difficulté par intuitivité. Dans les tutoriels nous introduisons les mécaniques et dynamiques complexes et peu intuitive en réduisant le nombre d'opportunité afin de rendre ces dynamiques plus facilement compréhensibles car plus intuitives.

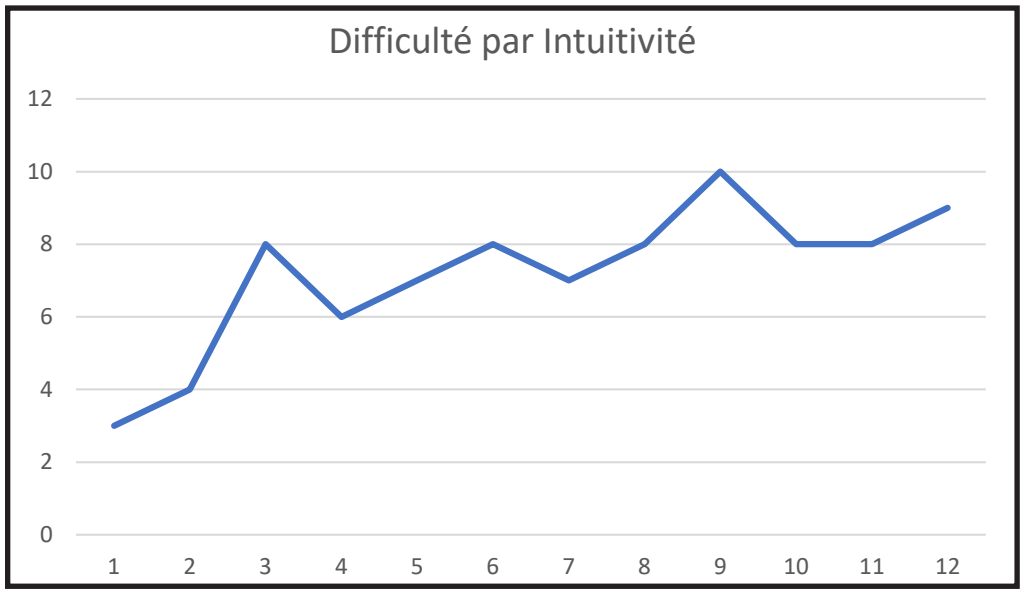
World	2	2	2	2	2	2	2	2	2	2	2	2
Niveaux	1	3	3	3	4	4	6	6	8	8	9	9
Situations	1	2	3	4	5	6	7	8	9	10	11	12
Difficulte par nombre d opportunite	1	2	4	2	6	6	2	8	10	6	6	10
Difficulte par Intuitivite	3	6	8	6	11	8	6	11	11	10	8	14
Difficulté par visibilité	3	6	4	0	0	2	0	2	0	5	0	0
Difficulte combinee	7	14	16	8	17	16	8	21	21	21	14	24
Difficulté combinée par niveau	7			38		33		29		42		38
Temps de completion moyen /s	5	5	6	3	5	6	4	8	12	5	8	14
Temps de completion total par niveaux /s	5			14		11		12		17		22

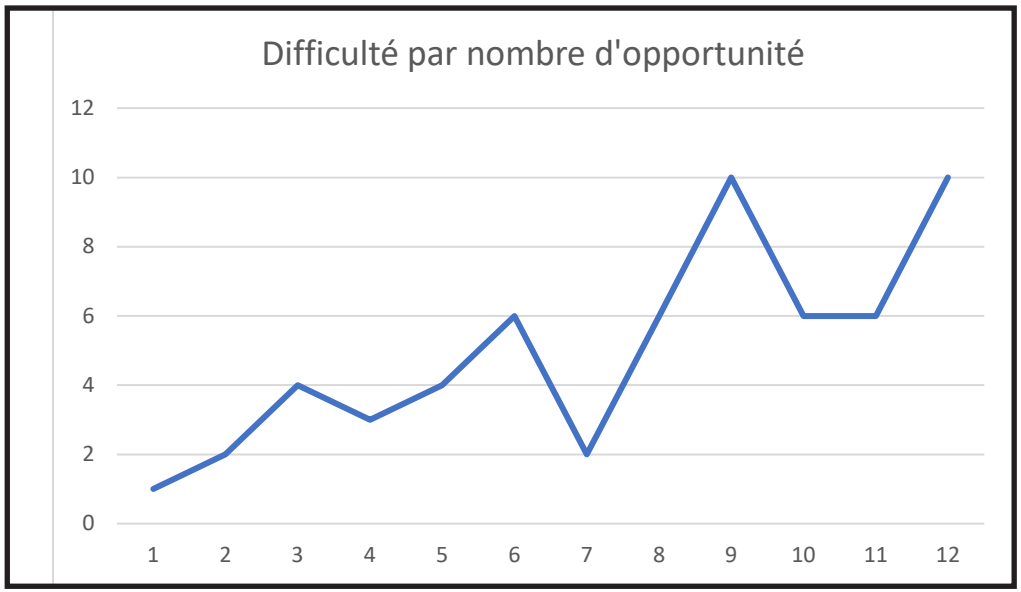
Rationnal Level Design

Courbes

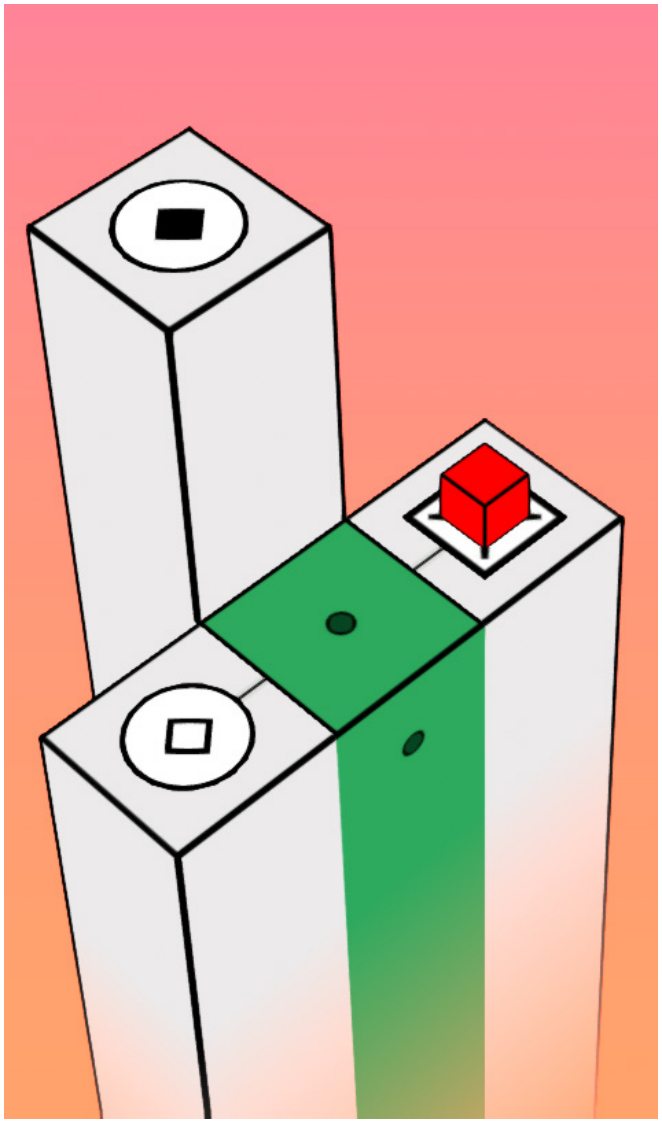






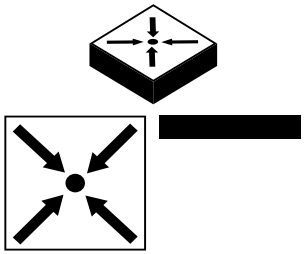


Niveau 2 - 1



Ingrédients

Tempo Tile:
- Verte * 1



Bloc activable * 1

Intentions

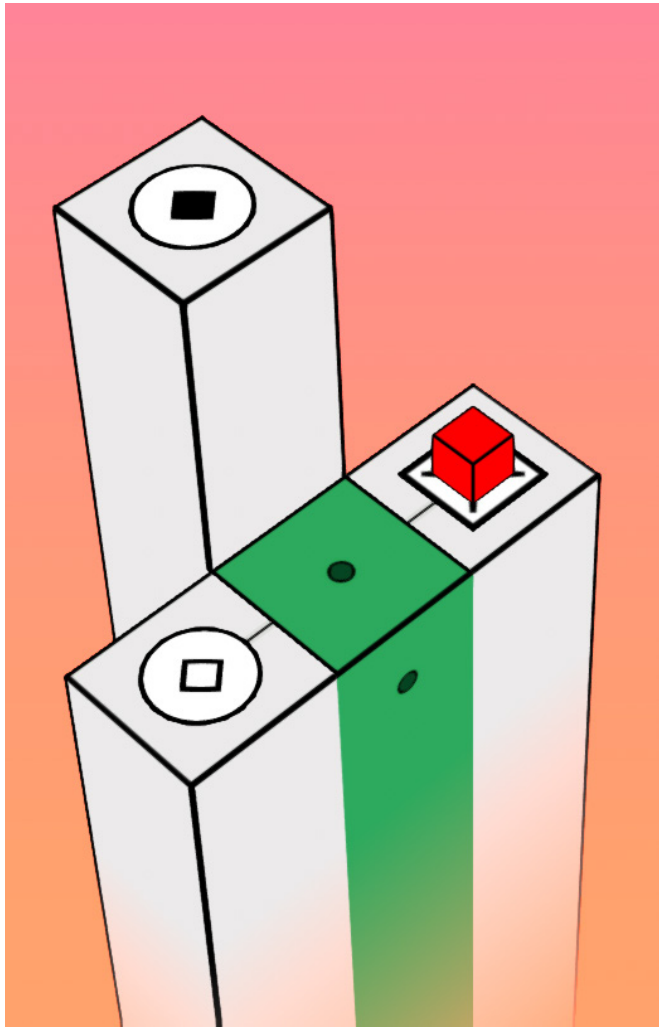
L'intention du premier niveau du World 2 était d'introduire le joueur au nouveau bloc de Level Design « le bloc activable» et plus particulièrement à la synergie que celui ci pouvait avoir avec la Core mechanic de notrre jeu qui sont les Tempo Tiles.

Le niveaux est construit de manière très simple il est uniquement résolvable en un nombre de coup précis le système d'étoile ne fonctionnent donc pas ici (4 déplacements).

L'élément le plus important dans le design du niveau n'est pas d'amener le joueur à la réflexion mais à l'apprentissage du nouvelle mécanique et d'une nouvelle synergie entre deux bloc de Level Design qui sera ammené a recroisé dans les futurs niveaux, Il est donc très compliqué pour le joueur de se tromper sur ce niveau.

Niveau 2 - 1

Situations



Unique situation

Le joueur aperçoit ce nouveau bloc de level design qu'est le bloc activable et se déplace intuitivement vers lui. Une fois sa fonction révélée, faire apparaître et disparaitre des cases qui ne sont pas présentent au lencement du niveau, la fin du niveau se trouve a deux déplacements du joueur.

Difficulté par nombre d'opportunité : 1/10
Le joueur n'a accès qu'à une case tout au long du niveau (sauf au deuxième déplacement mais la marche arrière n'est ici pas intuive), son déplacement est même scripté pour l'aprentissage d'un nouveau comportement.

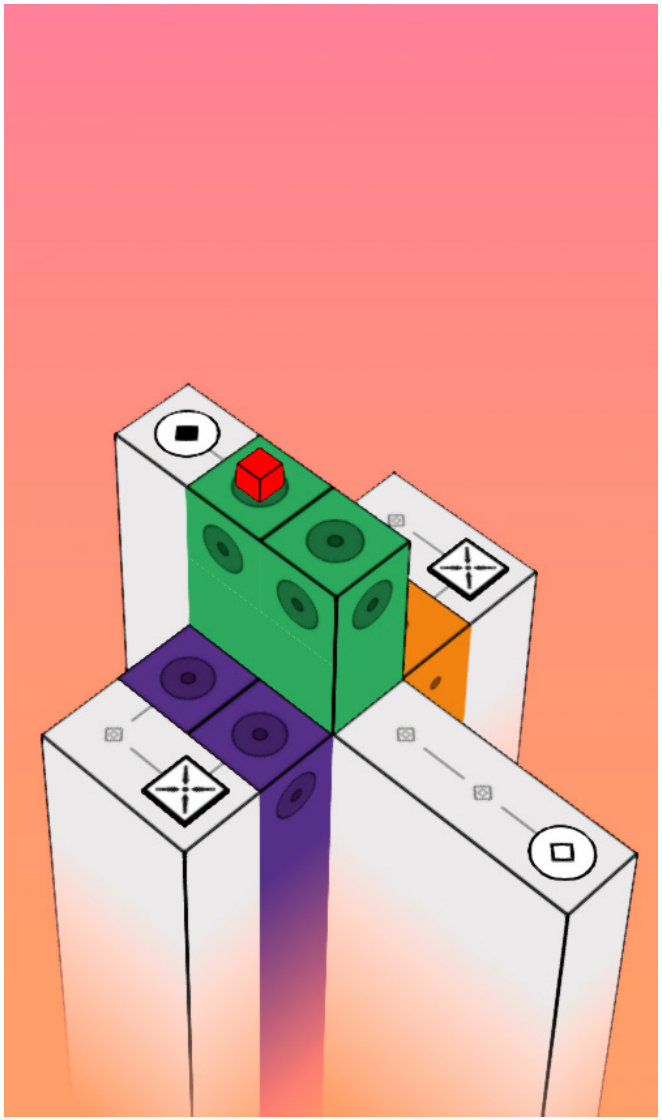
Difficulté par intuitivité : 3/10
La difficulté par intuitivité ici est intimement lié a la lisibilité du niveau. A part la méconaissance du comportement de ce nouveau bloc de level design, la difficulté par intuitivité malgré le peu d'opportunité de déplacement augemente.

Difficulté par visibilité : 3/10
Le comportement du bloc activable ne permet pas d'avoir une lecture totale sur les challenges mis en place par le niveau. Le joueur ne connait pas l'entièreté du niveau , il doit donc anticiper l'effet de la tile et s'adapter à ses conséquences.

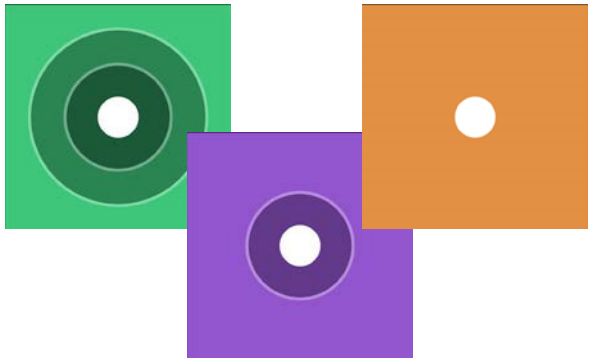
Difficulté combinée : 7

Rationnal Level Design

Niveau 2 - 3

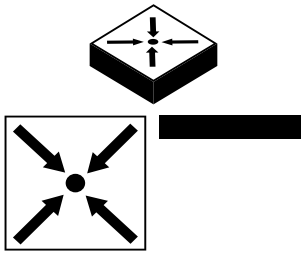


Ingrédients



Tempo Tile:
- Verte * 2
- Violette * 2
- Orange * 2

Bloc activable * 2



Intentions

Après avoir introduit le bloc activable sur les deux précedent niveaux du World 2, l'interet ici était de challenger le joueur avec des dynamiques plus complexes. Combiner les trois Tempo Tiles et leur synergie entre elles et les bloc activables creer de nouceaux challenges et pousse le joueur à réflechir d'avantage.

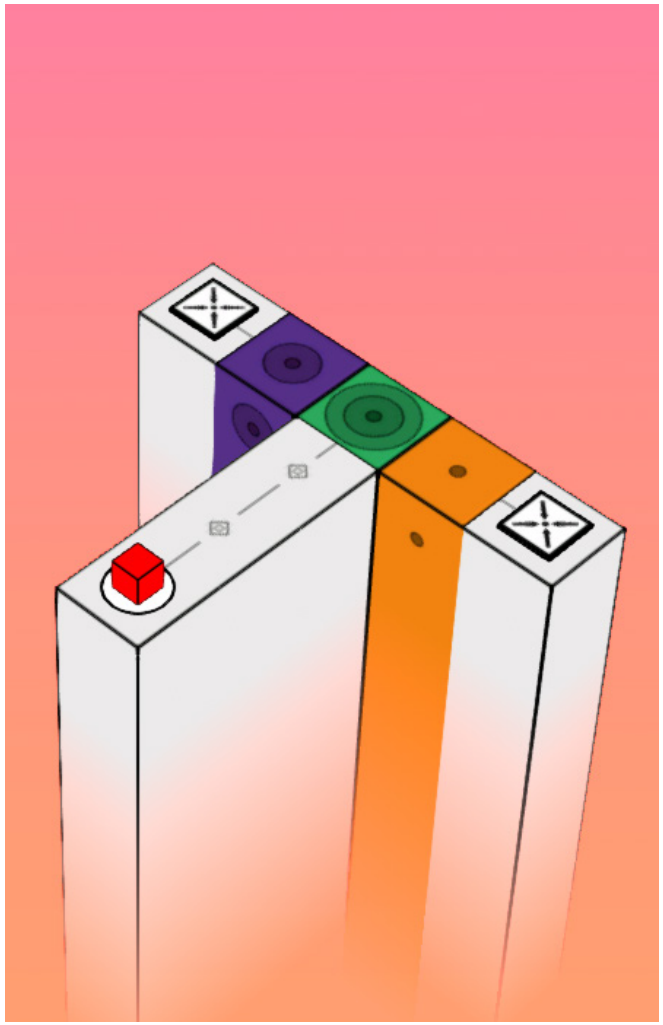
Le niveau est découpé en deux trois situatuins qui sont les trois game state du niveaux : atteindre le premier bloc activable voir comment il réagit, puis activer le deuxième et voir comment il réagit aussi et enfin atteindre la case de fin de niveau.

Le comportement des Tempo Tiles entres elles ici sert à frainer l'avancée du joueur entre les deux blocs activables ainsi qu'a fractionné le niveau en plusieurs situations.

Rationnal Level Design

Niveau 2 - 3

Situations



Première situation

Lorsque le joueur arrive dans le viveau il doit se rendre sur le bloc activable de droite , pour se faire il doit passer par les Tempo Tles verte et orange. En ce déplacant directement dessus son avancée n'est pas frainer et les Tempos Tiles lui ouvre naturellement un chemin.

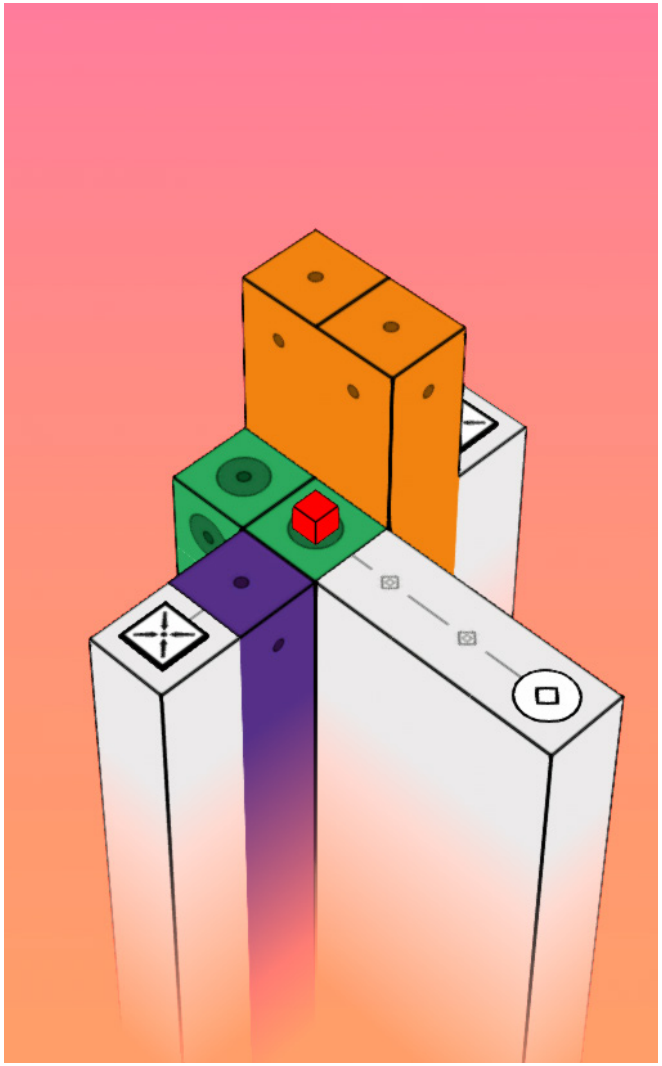
Difficulté par nombre d'opportunité : 2/10
Le joueur n' a pas beaucoup d'opportunité de déplacement car la Tempo Tile violette lui barre le chemin sur la gauche. A part si le joueur effectue une marche arrière, le déplacement du joueur ne nécéssite pas de calculs

Difficulté par intuitivité : 4/10
La difficulté par intuitivité ici est lié a la compréhension de son objectif a court terme . Le joueur ne sait pas si il faut se diriger vers le bloc activable de gauche ou celui de droite et si le comportement des Tempo Tiles va le frainer.

Difficulté par visibilité : 5/10
La difficulté par visibilité sur cette situation se traduit par le large espace potentiellement activable grâce au bloc et donc l'anticipation de ses prochains déplacements et par le fait que le joueur n'apercoive pas la case de fin de niveau.

Difficulté combinée : 11

Niveau 2 - 3



Deuxième situation

Le joueur doit maintenant traverser le niveau composé par plusieurs Tempo Tiles. Il doit désormais calculer ses futurs déplacements et utilisé les dynamiques propres au tris Tempo Tiles pour atteindre Le deuxième bloc Activable et donc la deuxième situation.

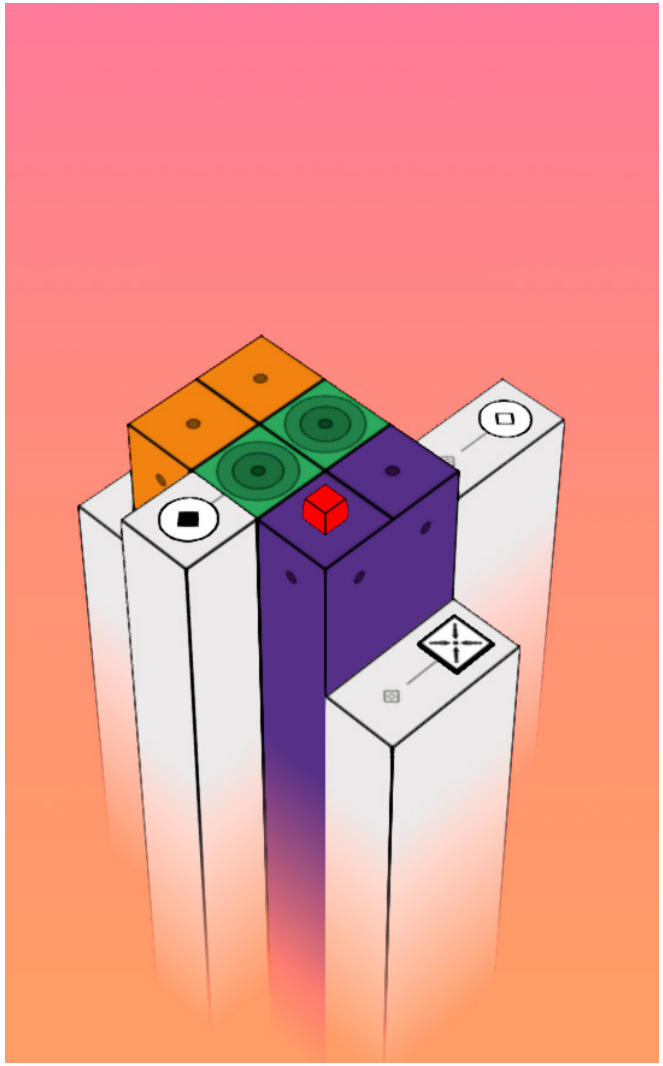
Difficulté par nombre d'opportunité : 4/10
La quantité de déplecement possible ici n'est pas énorme mais, chaque mauvais déplacement peut rapidement faire tourner le joueur en rond et lui faire acumuler un grand nombre déplacement.

Difficulté par intuitivité : 8/10
La difficulté par intuitivité se retrouve ici dans la capacité du joueur a calculer ses futurs déplacements et utilisé trois dynamiques différentes propres aux Tempo Tiles pour atteindre le deuxième bloc activable.

Difficulté par visibilité : 4/10
La difficulté par visibilité dans cette situation est similaire à la première mais légèrement moins forte car le joueur anticipe plus facilement la potentielle construction du niveau. Il peut donc mieux prévoir le cheminement jusqu'à la case de fin de niveau.

Difficulté combinée : 16

Niveau 2 - 3



Troisième situation

La fin du niveau est maintenant a quelques cases. Le joueur doit réutilisé les Tempo TIles pour s'y rendre et donc de nouveau calculer son déplacement.

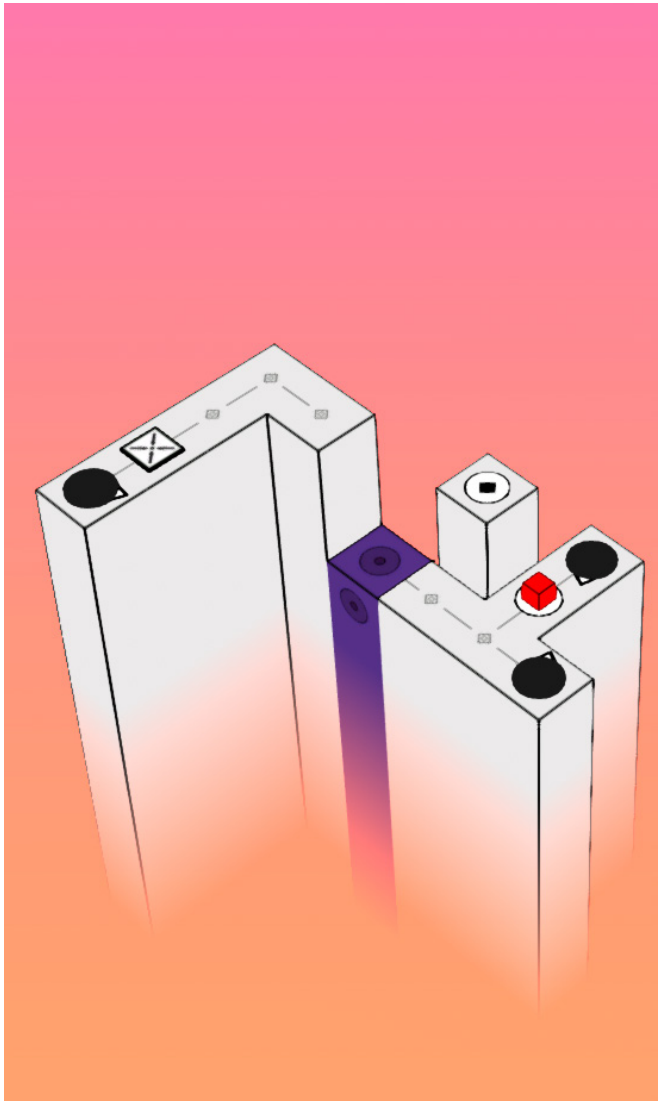
Difficulté par nombre d'opportunité : 3/10
La difficulté par nombre d'opportunité est ici en lien avec le comportement des Tempo Tiles, en effet les les Tempo Tiles et plus particulièrement les vertes forcent un passage pour le joueur et limitent donc les autres possibilités. (Par contre si le joueur fait le choix d'aller sur la case neutre à côté du bloc activable il se retrouve pris dans une boucle de 6 déplacement en plus)

Difficulté par intuitivité : 6/10
La difficulté par intuitivité sur la dernière situation est en relation avec le nombre d'opportunité. Les possibilités de déplacement étant restreintes, les potentiels calculs que le joueur aurait fait son justifier par un chemin presque unique.

Difficulté par visibilité : 0/10
La difficulté par visibilité est par nature lié a l'anticipation crée par le bloc activable. Les cases du niveaux étant toutes révélées il n'y a pas de difficulté lié a la lisibilité dans cette situation.

Difficulté combinée : 16 **Difficulté du niveau : 36**

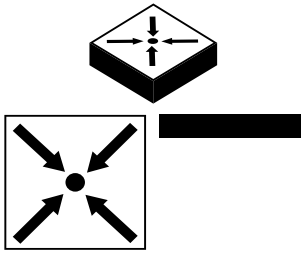
Niveau 2 - 4



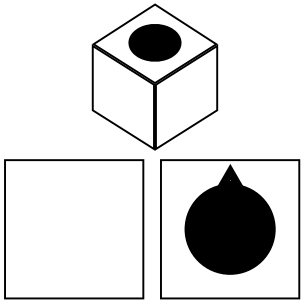
Ingrédients

Tempo Tile:
- Violette * 1

Bloc activable * 1



Bloc TP * 3



Intentions

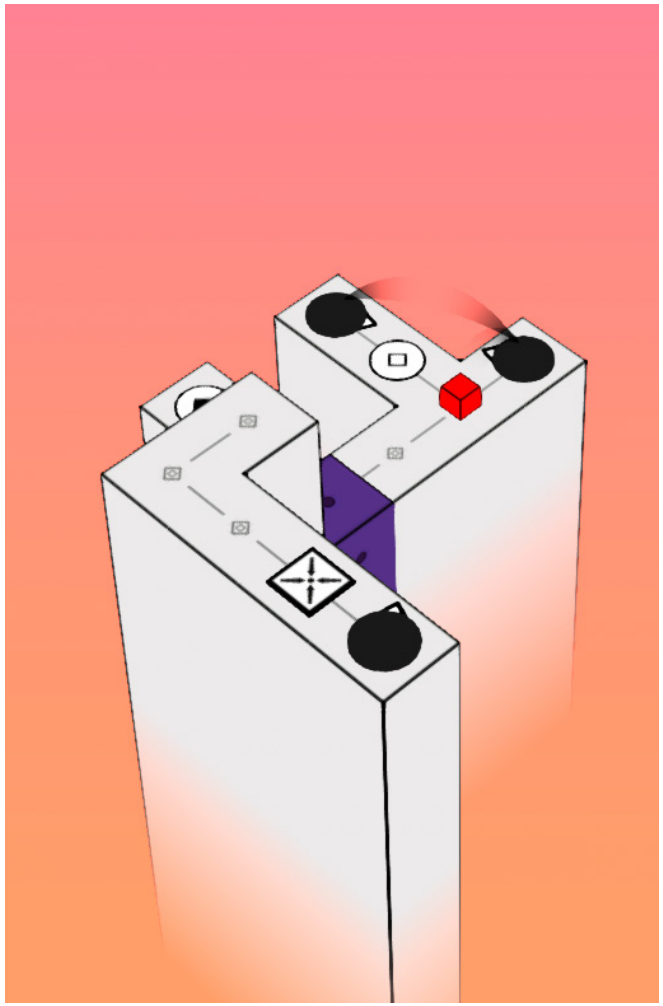
Après avoir introduit les blocs activables avec les Tempo Tiles et donc créer de nouvelles synergies, nous voulions avoir un niveau construit avec trois de nos blocs de Level Design.

Ici le bloc de téléportation «TP», créer une nouvelle dynamique avec le bloc activable. Les deux mêlés a la Tempo Tile violette découpe le niveau en deux partie : la partie basse et la partie haute. Les deux partie sont jointe par les blocs TP et la Tempo Tile violette.

Une de nos intentions était aussi que le niveau soit aéré de manière à créer de la distance entre les blocs. Cela pousse le joueur au calcul et à ce mettre «off-tempo».

Niveau 2 - 4

Situations



Première situation

Le joueur doit atteindre le bloc activable sur la partie haute en s'aidant de la Tempo Tile violette, pour ce faire il doit se mettre sur un autre tempo avec les blocs de téléportation.

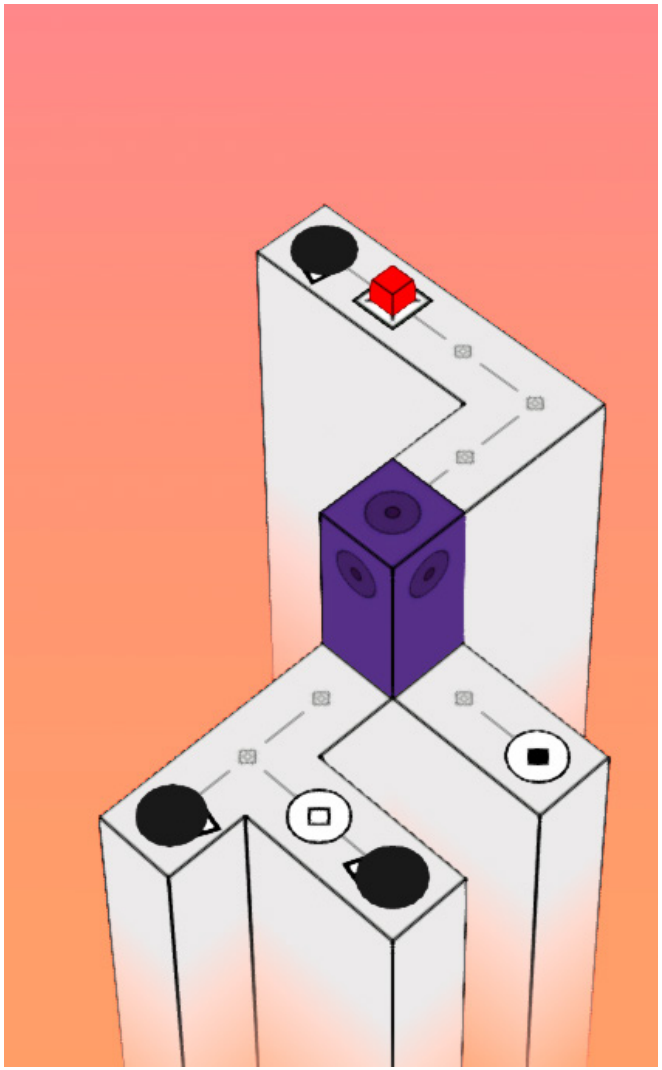
Difficulté par nombre d'opportunité : 4/10
Les deux blocs TP permettent au joueur de se caler sur le bon tempo de la Tempo Tile violette mais l'embranchement des deux chemins menant a la tile violette determine si le joueur doit faire une marche arrière ou non. Le nombre d'oportunité n'est pas grand mais le choix de plusieurs chemins l'est.

Difficulté par intuitivité : 7/10
La difficulté par intuitivité ici est influancé par l'angle pris par la caméra, le joueur sous l'angle par défaut de la caméra est plus amené a se dirigé directement vers l'unique Tempo Tile alors qu'il doit se servir de la dynamique de changement de tempo des blocs TP.

Difficulté par visibilité : 3/10
La case de fin est séparé d'une seule case et n'est pas accésible au début, le joueur peut donc facilement s'attendre à l'effet du bloc activable.

Difficulté combinée : 14

Niveau 2 - 4



Deuxième situation

Le joueur doit maintenant se diriger vers la case de fin de niveau, il peut y accéder de deux manière différente : soit en rebroussant chemin en 18 coups, soit en s'aident des bloc TP mais il utilisera alors 19 coups.

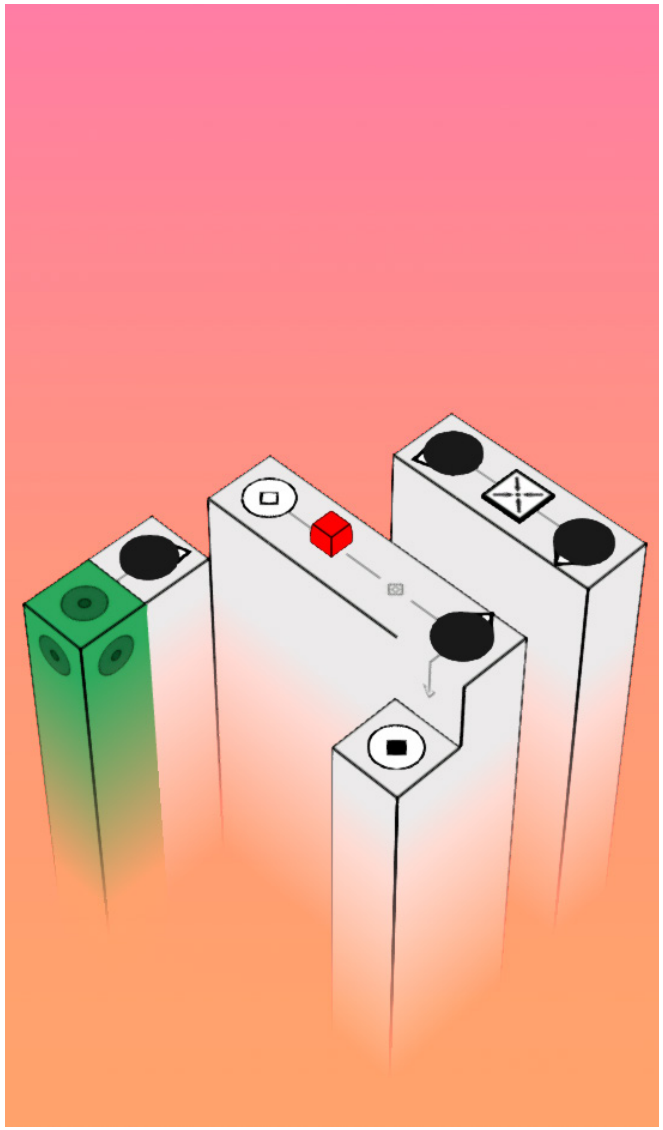
Difficulté par nombre d'opportunité : 6/10
Tout comme la première situation avec un embranchement de chemins, la difficulté par nombre d'opportunité n'est pas défini ici par le nombre de possibilités de chemins mais par le choix d'un chemin plus ou moins long.

Difficulté par intuitivité : 8/10
La difficulté par intuitivité dans cette situation est en lien avec le calcul du nombre de déplacement optimale et la sens du parcours déja fait par le joueur. Si le joueur prends le TP ce vers quoi il se dirige naturellement il se confronte a de nouveaux calculs de déplacement.

Difficulté par visibilité : 0/10
La difficulté par visibilité est par nature lié a l'anticipation crée par le bloc activable. Les cases du niveaux étant toutes révélées il n'y a pas de difficulté lié a la lisibilité dans cette situation.

Difficulté combinée : 14 **Difficulté du niveau : 28**

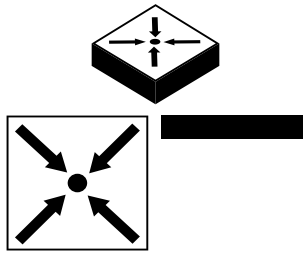
Niveau 2 - 6



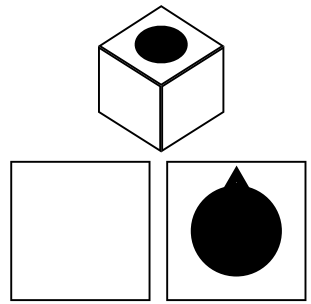
Ingrédients

Tempo Tile:
- Verte * 1

Bloc activable * 1



Bloc TP * 4



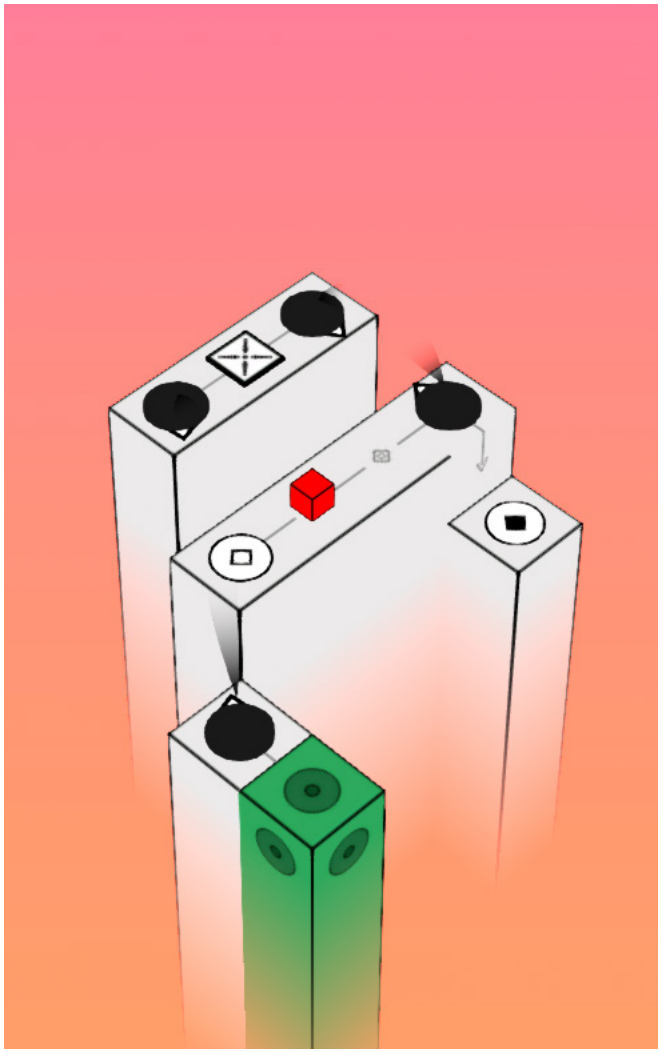
Intentions

L'intention sur le niveau 2 - 6 du World 2 était de pousser l'aspect programmation un peu plus loin que d'habitude en demandant au joueur de faire un déplacement particulier en début de niveaux qui aura un impact plus tard dans le niveau.

Le niveau est construit de telle sorte a ce qu'on distingue trois parties : la partie basse, la partie moyenne et la partie haute. C'est trois partie ne sont pas relié au lancement de la partie mais elle finissent par se rejoindre lorsqu'on fait usage du bloc activable.

Niveau 2 - 6

Situations



Première situation

Le joueur doit atteindre la Tile verte, pour se faire il doit effectuer une marche arrière avant de prendre le bloc TP. En plus de cela, le bloc activable en plus d'activer des cases désactive le TP permettant d'atteindre la case de fin de niveau rapidement.

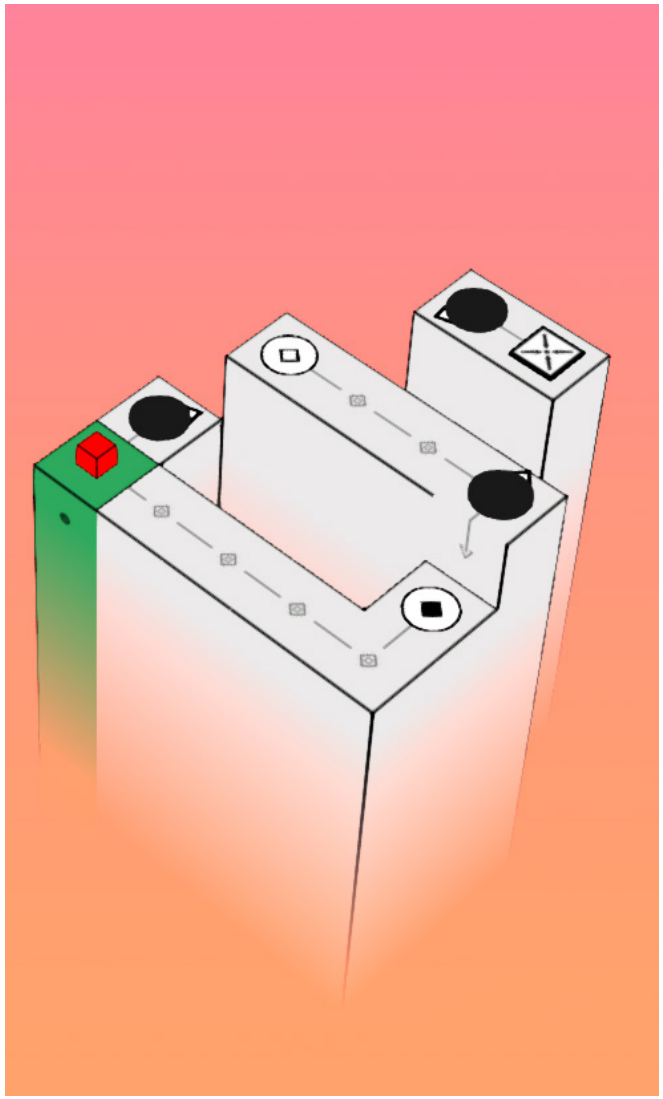
Difficulté par nombre d'opportunité : 2/10
Le déplacement dans cette situation est presque scripté le joueur ne peux qu'avancer ou reculer, ajoutez à cela que le bloc activable empechent le joir de faire marche arrière après ca, nous pourrions même affirmer que le chemin emprunter par le joueur est forcé.

Difficulté par intuitivité : 7/10
La difficulté par intuitivité est défini dans ce niveau et dans cette situation en particulier par le fait qu'il est très peu intuitif pour le joueur de faire une marche arrière pour accéder à la Tile verte plus tard dans le niveau. Alors que c'est obligatoire pour progresser dans le niveau.

Difficulté par visibilité : 4/10
La difficulté par visibilité sur cette situation se traduit par un grand espace vide mais potentiellement activable grâce au bloc et donc l'anticipation de ses prochains déplacements et par le fait que le joueur n'apercoive pas la case de fin de niveau.

Difficulté combinée : 13

Niveau 2 - 6



Deuxième situation

Le joueur fait maintenant face a deux choix : un très peu intuitif ou il faut revenir sur ses pas et sur lequel on a que très peu de visibilité quant au chemin que l'on a souhaiter prendre et le deuxième est un chemin qui fait sens avec le parcours initial du joueur mais qui utilise plus de coup de déplacement.

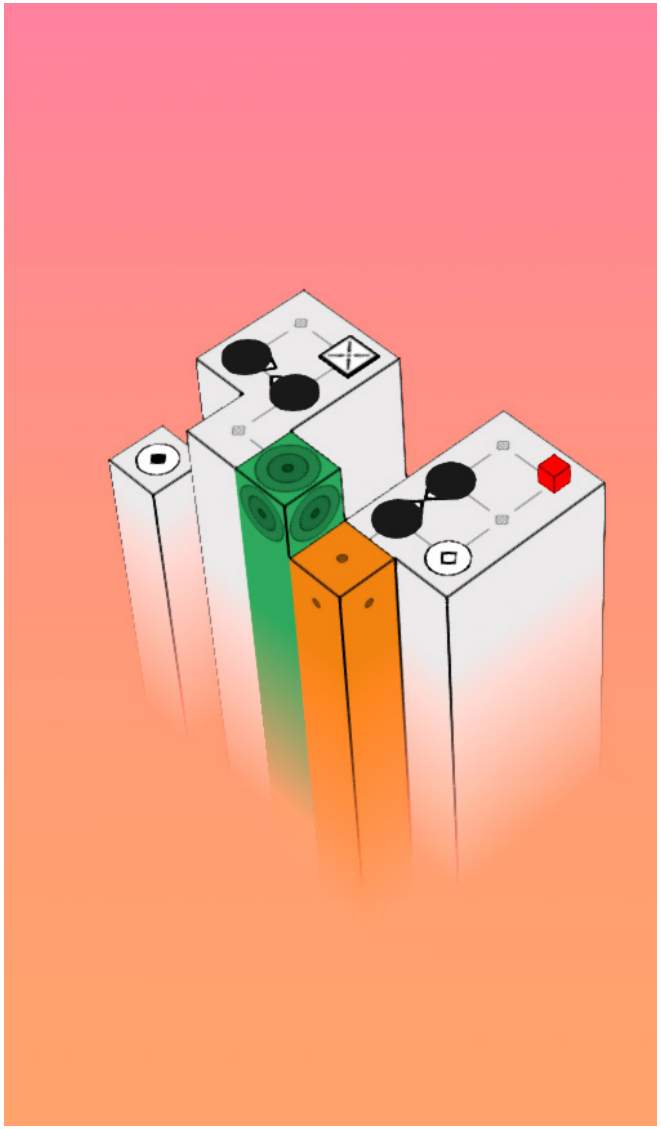
Difficulté par nombre d'opportunité : 6/10
La difficulté par opportunité est induite par le choix d'un des deux chemins et même si le parcours effectué par le joueur semble limité car linéaire, il est en réalité plus complexe car lié a l'intuitivité.

Difficulté par intuitivité : 8/10
Comme dit précedemment, il est très peu intuitif pour le joueur de revenir sur ses pas (surtout deux fois dzans le même niveaux) pour faire le chemin optimale, ajoutez a cela que le deuxième choix plus intuitif parait simple et rapide alors que le premier n'est pas totalement visible.

Difficulté par visibilité : 2/10
La difficulté par visibilité est lié dans cette situation au choix de parcours le moins intuitif, c'est à dire celui ou on revient sur le TP, il faut utilisé de sa mémoire pour comprendre que l'on peut accéder à la case de fin de niveau par les deux premiers blocs TP mais en les utilisant dans le sens inverse.

Difficulté combinée : 11 **Difficulté du niveau : 29**

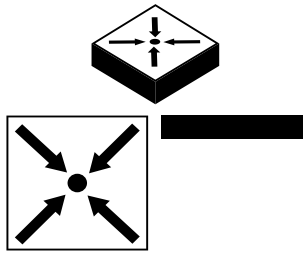
Niveau 2 - 8



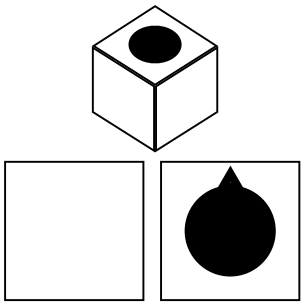
Ingrédients

Tempo Tile:
- Verte * 1
-Orange * 1

Bloc activable * 1



Bloc TP * 4



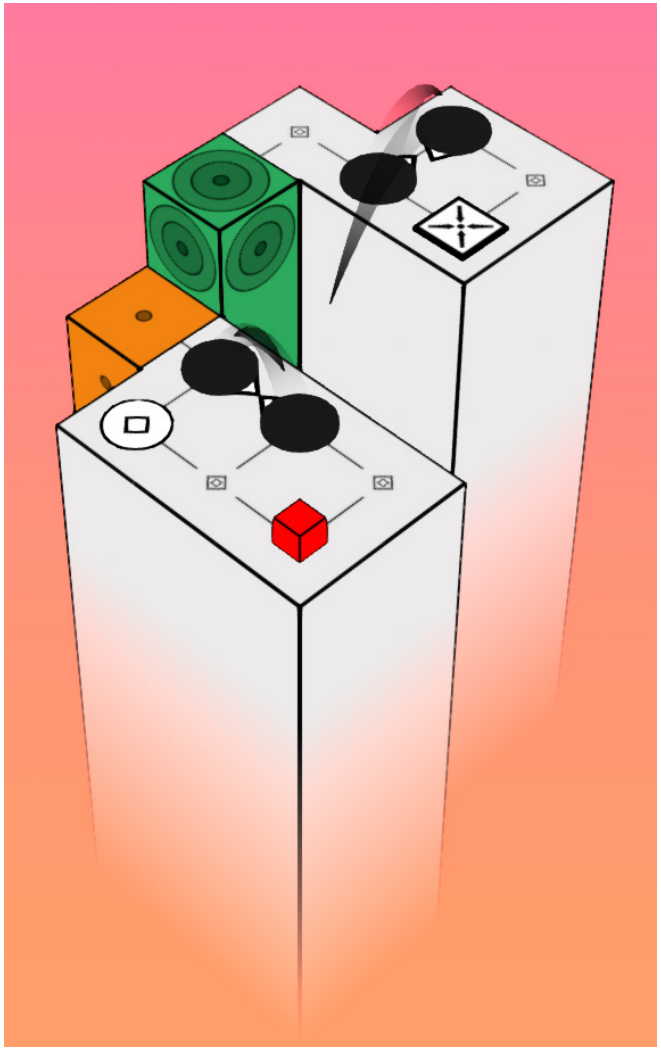
Intentions

L'intention principale lors de la création de ce niveau était une punitivité assez forte, pas comme une défaite car il n'y a pas de condition de défaite dans Araka mais par l'écart colossal du nombre de déplacement entre le nombre de déplacement optimal c'est à dire les trois étoiles et le nombre de déplacement medium (deux étoiles).

Le niveau est donc construit de tel sorte a ce que le joueur, s'il ne choisi pas le bon déplacement au bon moment recommence un challenge précedent et donc accumule beaucoup de nombre de déplacement, pour se faire le niveau est construit en deux parties relié par une combinaison Tempo Tile verte et orange. Chaque parties représente un challenge et plus ou moins une situation.

Niveau 2 - 8

Situations



Première situation

Le joueur doit atteindre le bloc activable sur la partie haute du niveau. Pour ce faire il doit se mettre dans un tempo synchronisé avec la Tempo Tile orange et la Tempo Tile verte en utilisant les blocs TP.

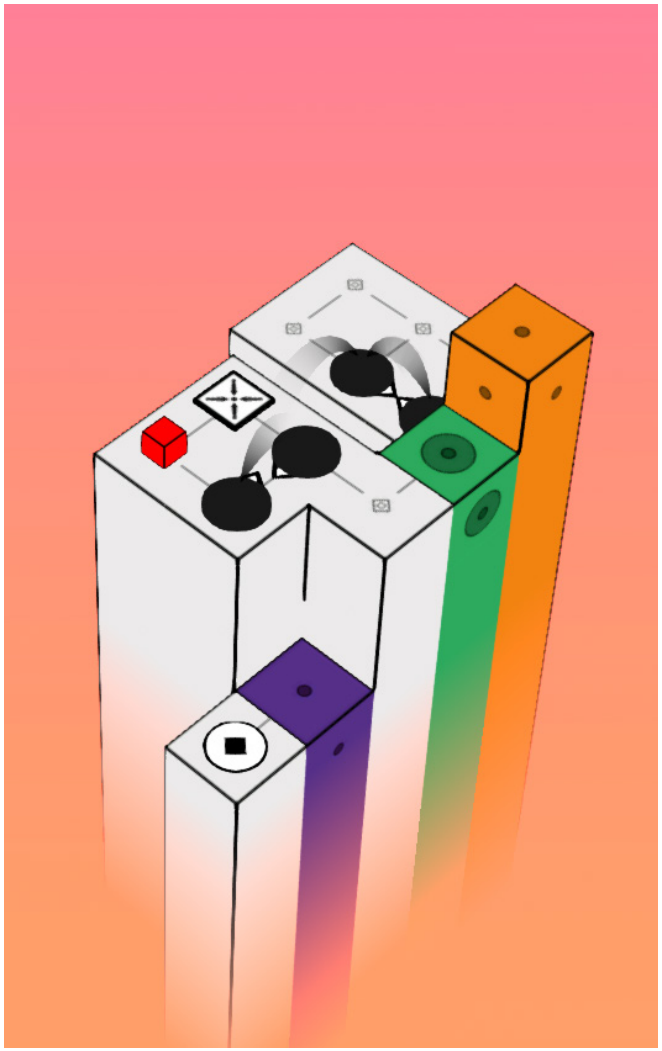
Difficulté par nombre d'opportunité : 10/10
La difficulté par nombre d'opportunité est accrue ici car l'espace du bas permet pleins de combinaisons de déplacement.

Difficulté par intuitivité : 10/10
Pour accéder à la phase du haut il faut, comme dit précedemment, se mettre en tempo synchronisé avec la tempo Tile orange et la verte. Pour se faire le joueur doit effectué un nombre et un shéma de déplacement très peu intuitif. C'est à dire cinq déplacement libres sur les cases neutres pour ensuite rejoindre le deuxième bloc TP.

Difficulté par visibilité : 3/10
La case de fin est séparé d'une seule case et n'est pas accésible au début, le joueur peut donc facilement s'attendre à l'effet du bloc activable.

Difficulté combinée : 23

Niveau 2 - 8



Deuxième situation

Le joueur doit desormais atteindre la case de fin de niveau, ildoit s'aider de la proximité des deux TP pour pouvoir être sur le bon tempo de Tempo Tile violette et donc descendre et accéder à la case de fin de niveau.

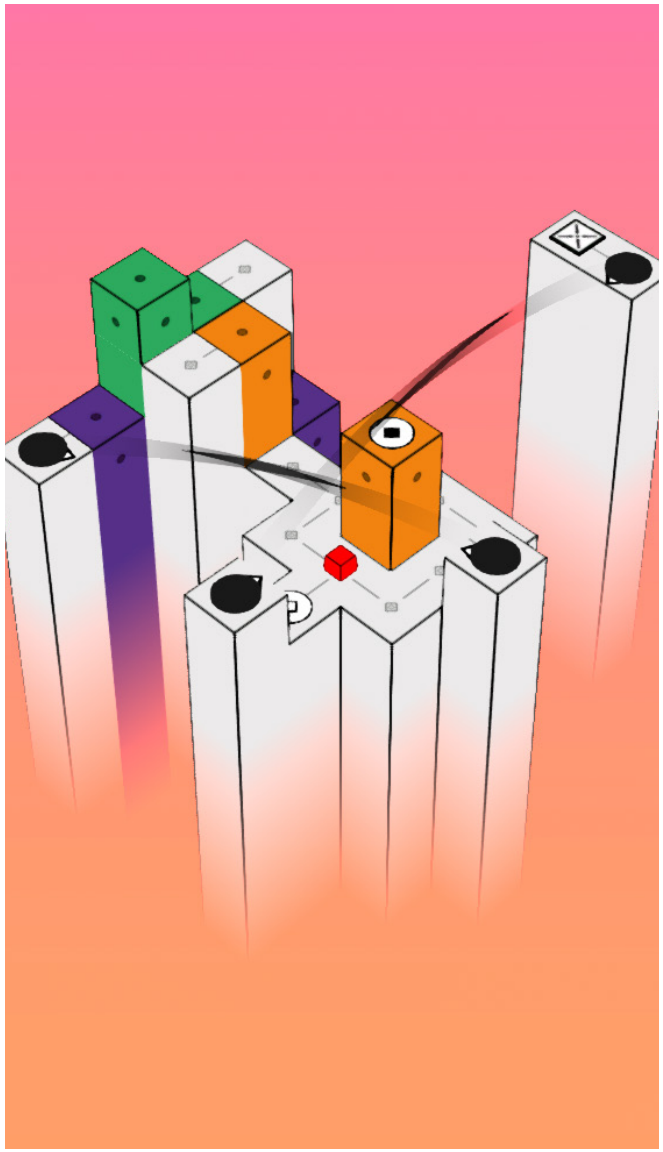
Difficulté par nombre d'opportunité : 6/10
La difficulté par le nombre d'opportunité est lié au fait que le mauvais choix de déplacement entraîne la punitivité du niveau et par conséquent le retour au premier challenges.

Difficulté par intuitivité : 8/10
Dans cette situation le joueur doit se mettre sur le TP du bas sans rentrer dedans sinon il retombe au début du niveau. Ensuite de manière très peu intuitive il doit sauter en boucle dans le bloc TP adjacent pour se mettre sur le bon rythme avec la Tempo Tile violette.

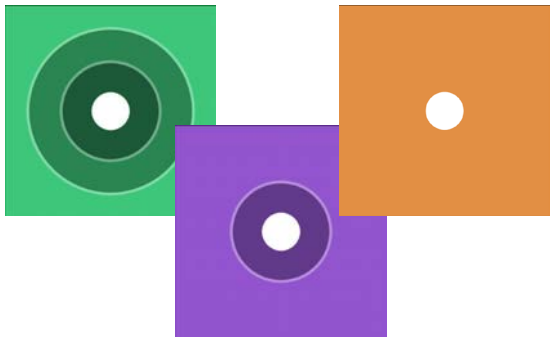
Difficulté par visibilité : 0/10
La difficulté par visibilité est par nature lié a l'anticipation crée par le bloc activable. Les cases du niveaux étant toutes révélées il n'y a pas de difficulté lié a la lisibilité dans cette situation.

Difficulté combinée : 14 **Difficulté du niveau : 37**

Niveau 2 - 9

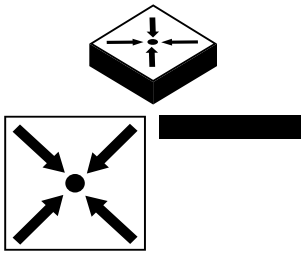


Ingrédients

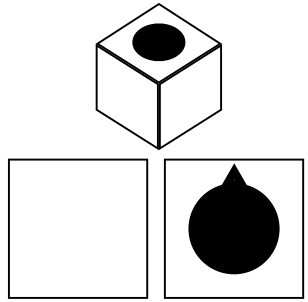


Tempo Tile:
- Verte * 2
- Violette * 2
- Orange * 2

Bloc activable * 1



Bloc TP * 4



Intentions

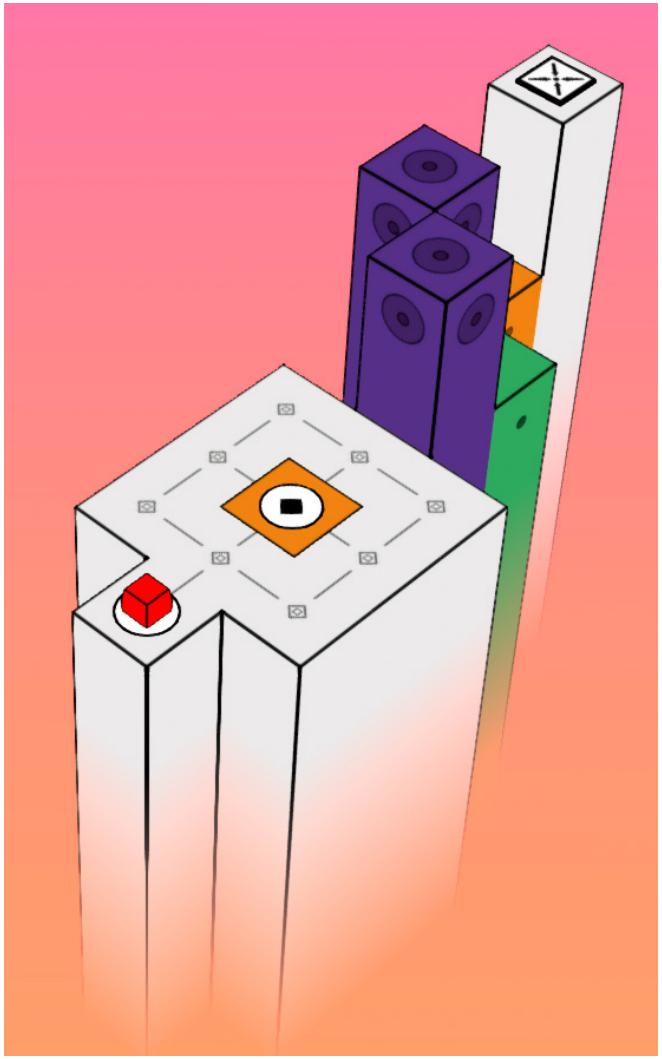
Le niveau est découpé de tel sorte a ce qu'il y est comme des petits niveaux dans le niveaux avec un hub centrale sur lequel se trouve la case de fin de niveau.

La proximité entre la case de fin de niveau et la case de départ était aussi une de nos intentions principale sur le niveau 2 - 9 car cela rajoute une frustration chez le joueur car il se sent si près d'atteindre son but alors qu'il ne peux pas.

L'enjeux pricipale est de ce mettre en tempo synchronisé avec la Tempo Tile orange de fin de niveau. Pour ce faire le joueur doit atteindre un bloc TP sur la deuxième partie du niveau.

Niveau 2 - 9

Situations



Première situation

Le joueur doit atteindre le bloc activable sur la partie haute de droite afin de dévoilé la partie gauche. Il devra passer par une succession de Tempo Tiles violette verte et orange en utilisant leurs synergies.

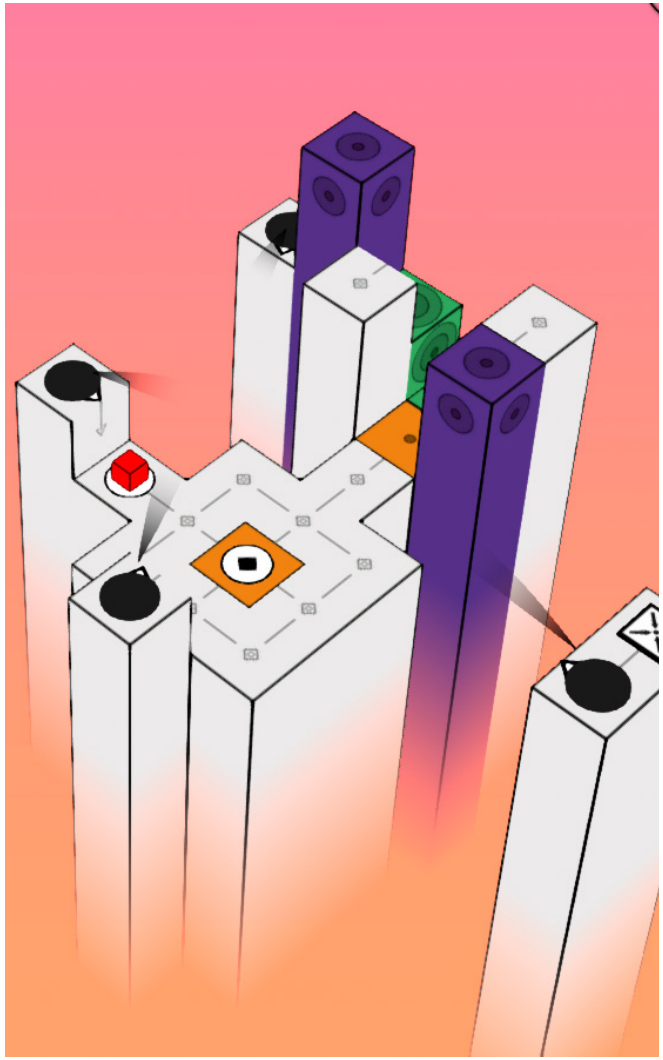
Difficulté par nombre d'opportunité : 6/10
La quantité de déplacement possible ici n'est pas énorme mais, chaque mauvais déplacement peut rapidement entraîner un blocage et l'utilisation du rewind et donc implémenté un ou plusieurs déplacement pas voulu au conteur.

Difficulté par intuitivité : 8/10
La difficulté par intuitivité se retrouve ici dans la capacité du joueur a calculer ses futurs déplacements et utilisé trois dynamiques différentes propres aux Tempo Tiles pour atteindre le premier bloc activable.

Difficulté par visibilité : 5/10
La difficulté par visibilité sur cette situation se traduit par le large espace potentiellement activable grâce au bloc et donc l'anticipation de ses prochains déplacements et par le fait que le joueur ne sache toujours pas comment et ou va être placé le bloc TP capable de le mettre en asyncrone.

Difficulté combinée : 19

Niveau 2 - 9



Deuxième situation

Le joueur doit atteindre le bloc TP qui vient d'apparaître pour atteindre la case de fin de niveau. Il doit passer par une partie constitué des trois type de Tempo Tiles jouant sur la verticalité créant beaucoup de situation de blocage.

Difficulté par nombre d'opportunité : 10/10
La quantité de choix de déplacement est bien plus élevé que sur les niveaux précedent. Le joueur effectue d'abord une phase de déplacement linéaire pour se rendre vers les Tempo Tiles puis une phase de réflexion et calculs sur ces choix de déplacement.

Difficulté par intuitivité : 9/10
La difficulté par intuitivité est induite par la quantité de possibilité de déplacement mais aussi par l'agencement des Tempo Tiles qui forcent la verticalité de cette situation la proximité entre les Tempo Tiles n'aide pas forcemment non plus à la bonne visibilité et aux calculs.

Difficulté par visibilité : 0/10
La difficulté par visibilité est par nature lié a l'anticipation crée par le bloc activable. Les cases du niveaux étant toutes révélées il n'y a pas de difficulté lié a la lisibilité dans cette situation.

Difficulté combinée : 19 **Difficulté du niveau : 39**

Programmation

- Structure de programmation -
- Enjeux majeurs -

Structure de Programmation

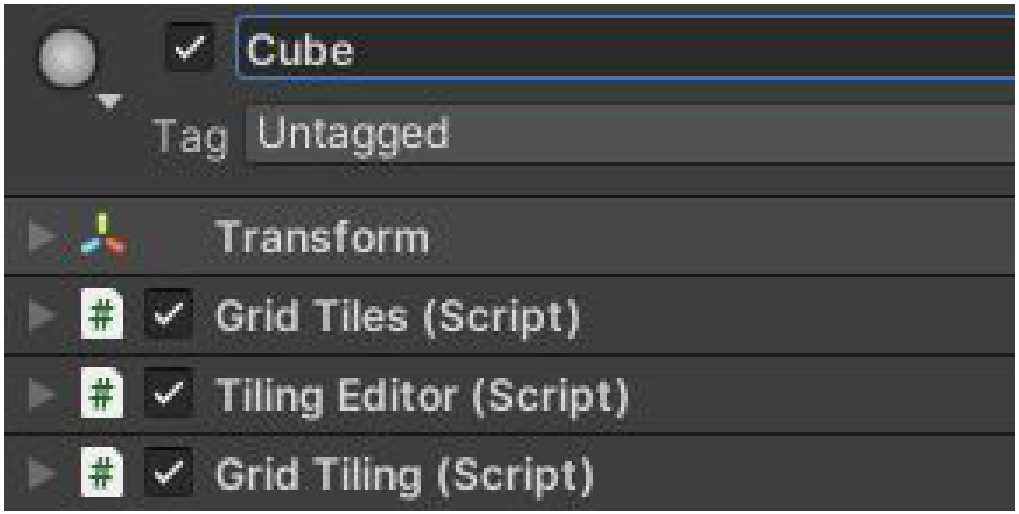
Les scripts dans Araka sont répartis en 4 catégories, «Major Grid and Game Systems», «Player and Camera», «State Machine» et «UI and Editing».

Major Grid And Game Systems

Chaques Tiles de notre Grille est dans un premier temps enregistré dans un tableau indexé par la position en X et en Z de chaque tile. Cette étape se déroule dans le «GridGenerator». Chaque tile possède 3 scripts principaux.

Un premier, «Grid Tiles» contenant les effets des blocs LD posés sur la tile ou du type de tile en question.

Le second, «Tiling Editor» contient les transformations et différents effets apparaissant directement dans l'éditeur *Unity* afin de rendre le level design reponsif.



Le troisième, «Grid Tiling» permet d'actualiser les materiaux des tiles régulièrement.

Le script «WorldManager» est un script uniquement utilisé dansle Hub pour lancer les différents niveaux.

Player and Camera

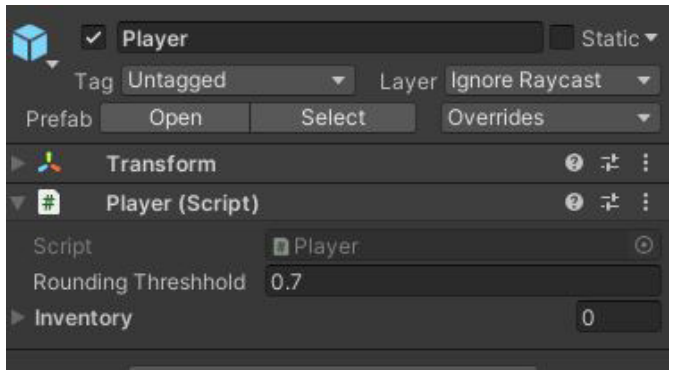
La catégorie Player and Camera contient les scripts traitant des 3C.

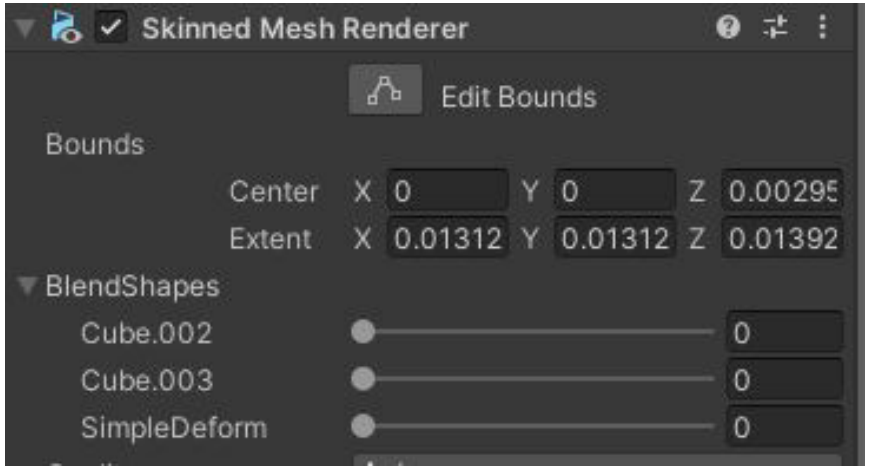
Le script «CameraBehavior» est un MonoBehavior placé sur le parent de la caméra appelé «Camera Container».

Ce Script contient la logique des différents déplacements et rotations de la caméra en «SmoothDamp» ainsi que du zoom et dézoom.

Le script «Player» Permet au joueur de toujours rester au dessus d'une tile et lance une partie des animations du joueur. Comme «Target», il est surtout utilisé pour référencer le joueur dans certains autres scripts grace à des «FindObjectOfType<Player>()».

Les animations du Player sont faites grace à des BlendShapeKeys présentes sur le Skinned Mesh Renderer dont on modifie la valeur pour distordre le joueur et créer des effets de squash rendant les déplacements juicy.



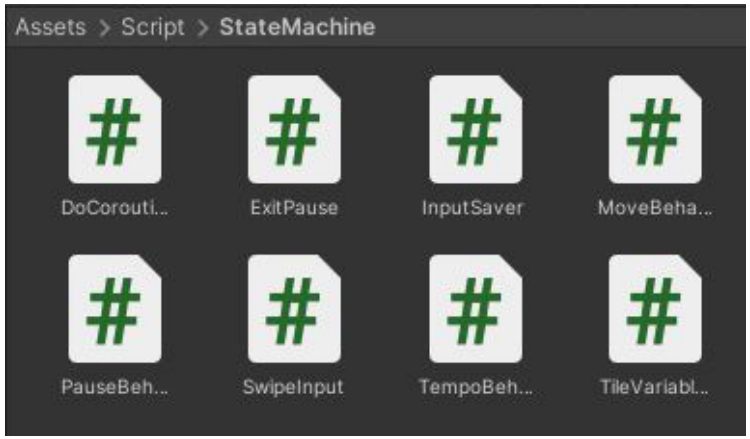


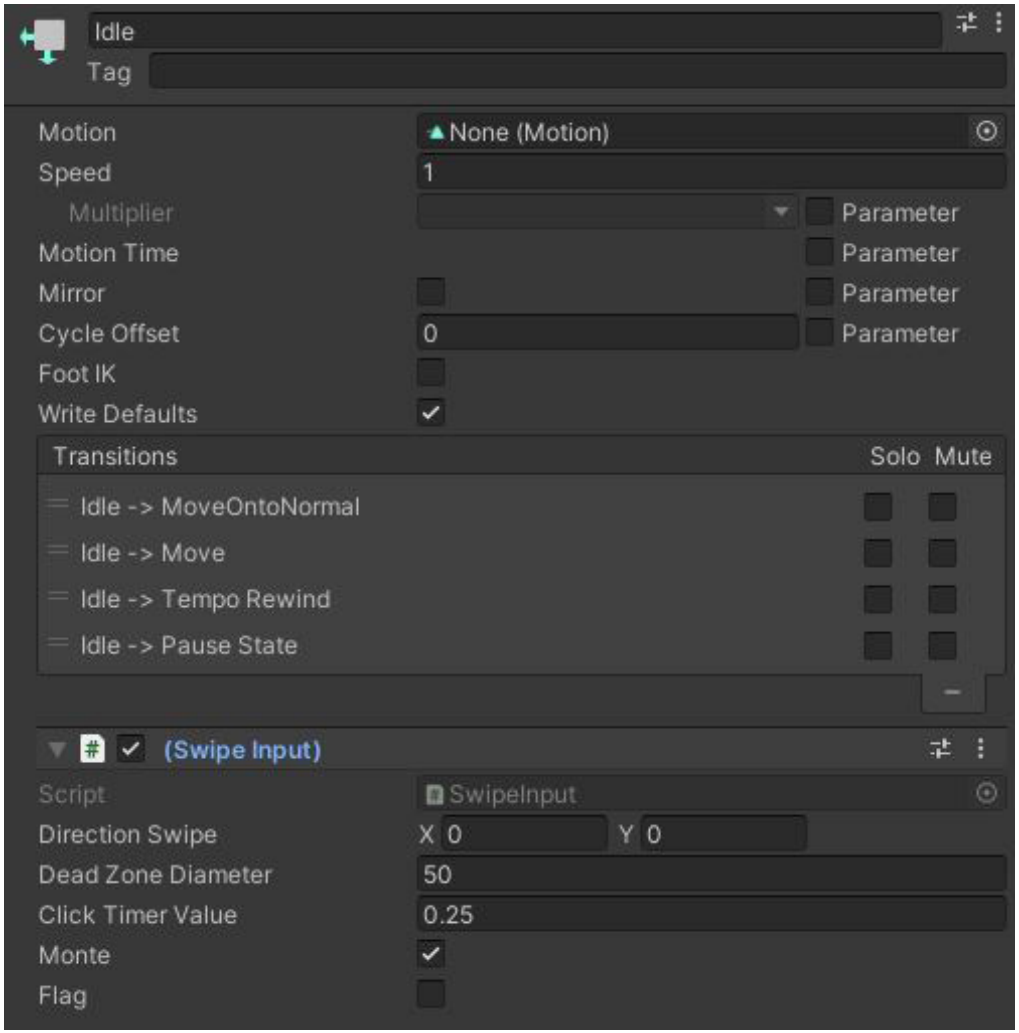
StateMachine

La StateMachine est un animator changeant d'état en fonctions des situations de jeu.

Elle est utilisée pour lancer les inputs du joueur, déplacer le joueur sur la grille, et actionner les déplacements des «tempo tiles». Elle est aussi utilisé pour les effets du Rewind et pour mettre le jeu en pause et le résumer.

Chaque état contient des scripts de type «StateBehavior» dont la logique est appliqué quand l'état est actif.



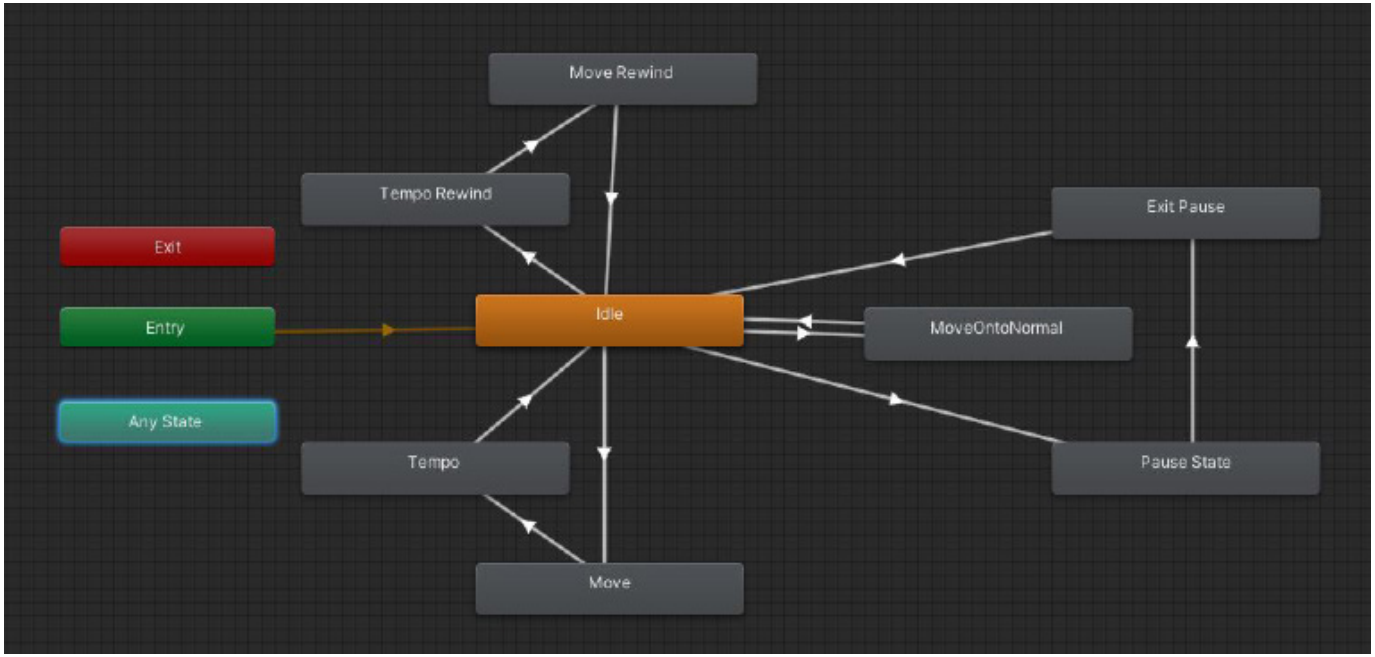


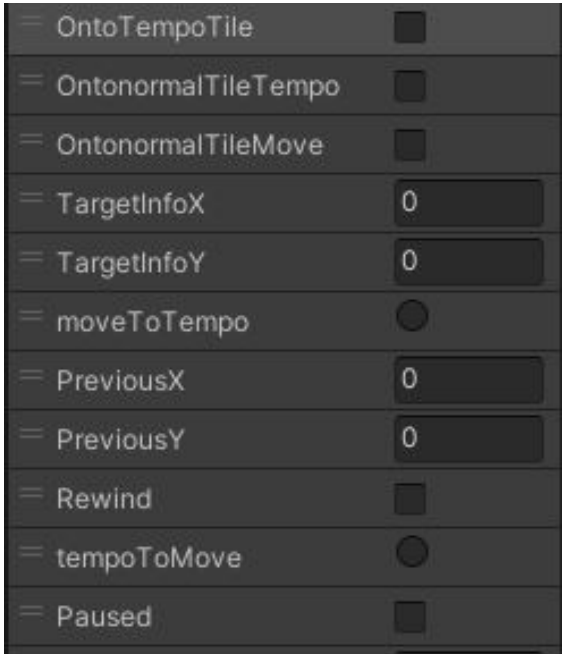
Structure de Programmation

La state Machine contient des paramètres d'animator permettant de créer des transitions entre les états à l'interieur de scripts State Behavior.

Ces paramètres sont aussi utiles pour enregistrer des valeurs importantes d'un état à un autre comme la position de la tile que le joueur quitte ou celle surlaquelle il va arriver («PreviousX, PreviousY» et «TargetInfoX, TargetInfoY»).

Les scripts StateBehavior de permettent pas d'utiliser les méthodes provenant des classes MonoBehaviors. Nous utilisons le Script nommé «DoCoroutine» ainsi que «InputSaver» pour utiliser les fonctions inhéritées du monobehavior comme «StartCoroutine». Ces scripts MonoBehavior sont placés sur le gameObject contenant l'animator de la stateMachine.





Structure de Programmation

UI and Editing

Les scripts de la catégorie UI and Editing sont ceux en rapport avec l'UI, la gestion de la scène, les changements de scènes et l'éditeur de niveau.

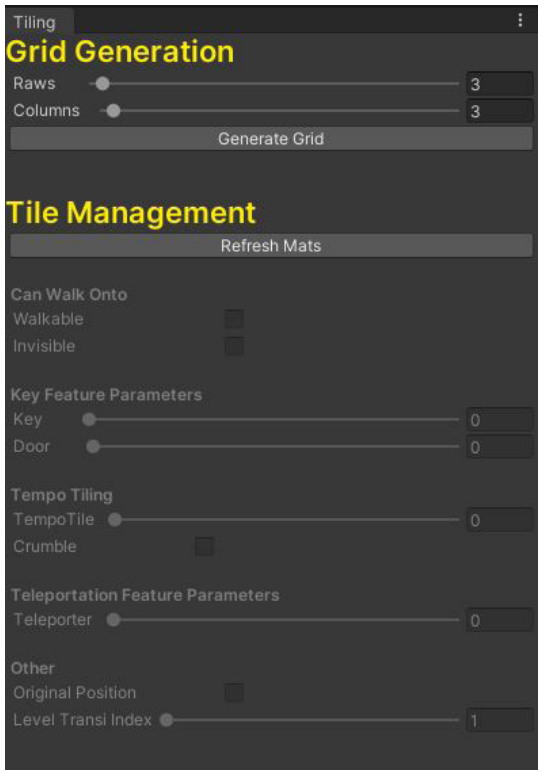
Le script «InGameUI» contient la majeur partie de la logique des différents boutons et images composant l'interface du joueur.

Le script «DebugTools» ainsi que le script «LoadScene» sont responsables des outils que nous utilisons dans le jeu pour faciliter les débuggage.

«Debug Tools» est aussi utilisé pour lancer la musique principale du jeu.

«SceneChange» est utilisé pour lancer les changements de scène dans le jeu et contient des méthodes utilisé pour lancer les transitions d'interface pour le menu de fin de niveau.

«TileEditorWindow» est un script de la classe «EditorWindow» qui créé une fenêtre nommé Tiling. cette fenêtre contient des boutons et curseurs permettant de faire du level design plus rapidement.



Agencement Intra-Script

Après avoir créé chaque script, nous avons pris un temps pour ranger les scripts et détailler dans l'inspecteur les variables à l'aide de 'notes', de 'headers', 'hide in inspector' et autres fonctions de rangement de Unity.

Ranger de manière optimale les scripts était important dans notre projet pour plusieurs raisons :

- Pour que les level designers apprennent facilement le rôle de chaque variable (détaillé dans 'Editeur' de la partie 'Enjeux majeurs de Programmation).

- Pour améliorer la clarté du code vis-à-vis des autres programmateurs.

- Pour faciliter les futures 'Bug Tracking' nécessitant de retourner sur d'anciens scripts.

```
public class PlayerMovement : MonoBehaviour
{
    [TextArea]
    [SerializeField] string Notes = "Comment Here.";

    #region variables

    [Header("Input Values")]
    [SerializeField] float moveSpeed;
    [HideInInspector] public Vector3 ogPos;
    public int currentPathIndex = 0;

    [Space]
    [Header("Components")]
    [SerializeField] StepAssignement stepAssignement;
    Reset reset;
    Transform player;


    [Space]
    [Header("Booleans")]
    public bool moveFlag;
    public bool moveState = false;

    [Space]
    [Header("Lists")]
    public List<GridTiles> highlightedTiles;
    GridTiles[,] grid;

    #endregion

    callMethods

    moveMethods

    TileBehavior
}
```

Enjeux majeurs

Enjeu de Génération de grille

Nous avions deux directions possibles pour la génération de la grille de chaque niveau :

- Générer en «Awake» la grille et associer à chaque point de la grille des coordonnées.

- Créer le level design du niveau en posant des tiles prefab en respectant des conventions précises (position dans l'espace en int, aucune position négative, aucune variance de rotation).

- Puis en Awake récupérer tous les cubes présents dans la scène dans un tableau puis les intégrer en indexé en fonction de leurs coordonnées 'x' et 'z' dans un autre tableau.

Nous avons choisi la seconde option pour faciliter le level design au prix d'un awake un peu plus chargé.

Enjeux de State Machine

Nous avons intégré plusieurs fonctions essentielles a notre jeu dans une strate machine utilisant comme base structurelle un animator. Cette strate machine contient l'input de déplacement, la logique de déplacement, la logique de mouvement de tile, l'effet du rewind et l'activation et désactivation du menu pause.

Optimisation de temps de déplacement

Notre déplacement s'effectue avant le mouvement des tiles. Cependant lorsque le joueur n'est pas sur une Tempotile ou n'arrive pas une Tempotile, attendre la fin du déplacement du joueur pour faire bouger les TempoTiles est une perte de temps qui peut frustrer le joueur. La state machine a donc deux comportements différents.

Si le joueur est sur une Tempotile ou arrive sur une Tempotile, le déplacement du joueur s'effectue avant le déplacement de la Tempotile. Dans le cas contraire le joueur et les TempoTiles se déplacent en même temps.

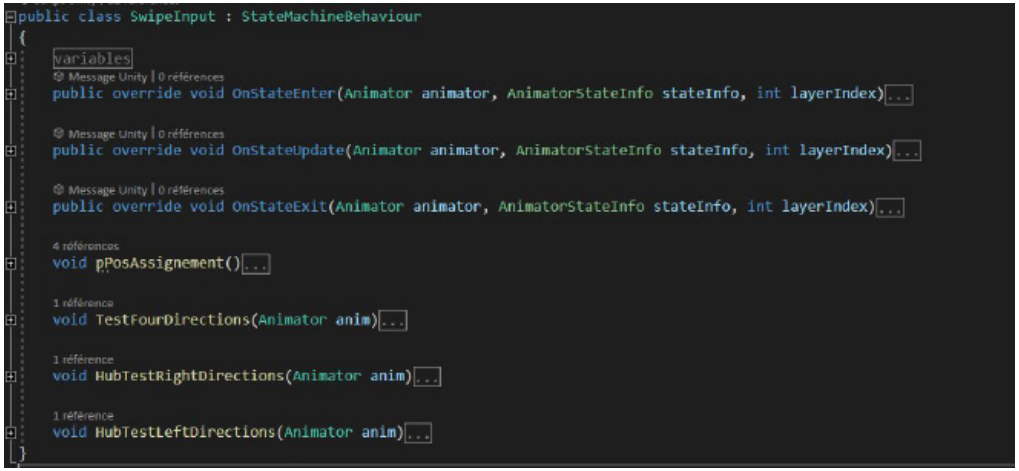
```
public class GridGenerator : MonoBehaviour
{
    [TextArea]
    [SerializeField] string Notes = "Comment Here.";
    #region variables
    public GridTiles[,] grid;

    [Header("Input Values")]

    public int raws;
    public int columns;

    #endregion
    Message Unity | 0 références
    void Awake()
    {

        GridTiles[] list = FindObjectsOfType<GridTiles>();
        grid = new GridTiles[raws, columns];
        for (int i = 0; i < list.Length; i++)
        {
            int x = (int)list[i].transform.position.x / (int)list[i].transform.localScale.x;
            int y = (int)list[i].transform.position.z / (int)list[i].transform.localScale.y;
            grid[x, y] = list[i];
            grid[x, y].name = "tiles " + x + " "+ y;
        }

    }
}
```

Enjeux majeurs

Swipe Input

Quand le joueur est en idle si jamais il swipe dans une direction la tile positionnée dans la direction vers laquelle le joueur a swipe est testée pour voir si elle est accessible. Si elle l'est, le code va tester si le joueur est positionné sur une Tempotile ou s'il arrivera sur une Tempotile.

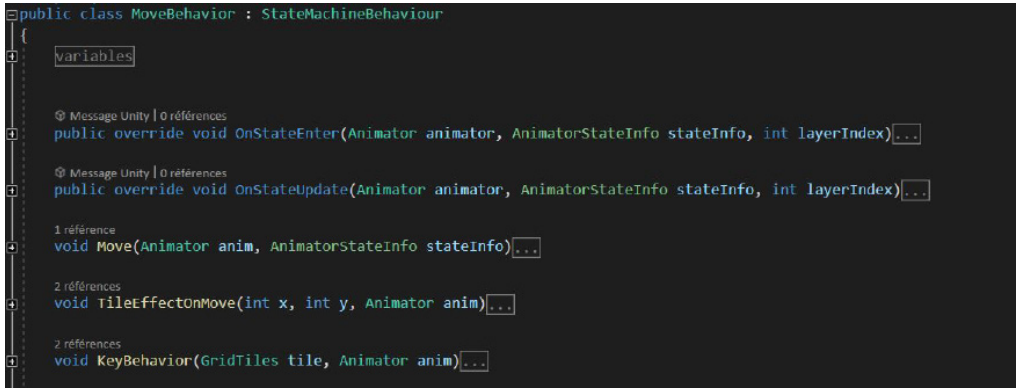
En fonction du résultat de ce test, l'un des 2 booléens de la state machine, « onToNormalTile» et «onToTempoTile» est activé et lance un autre State de l'animator.



MoveBehavior

Quand le joueur sort de idle, le joueur va se déplacer vers une case adjacente. le script MoveBehavior contient la logique de ce déplacement ainsi que les effets de tile activés lorsque le joueur arrive sur la tile adjacente.

À la fin du déplacement, la strate machine revient en idle si le TempoBehavior est fini ou passe au TempoBehaviors s'il n'a pas encore été lancé.



Enjeux majeurs

TempoBehavior

Le script de TempoBehavior contient la logique du déplacement des TempoTiles ainsi que de leur fonctionnement plus général. Si aucune tiles ne nécessite de mouvement après le déplacement du joueur ou si le mouvement est fini, la state machine repasse en idle.

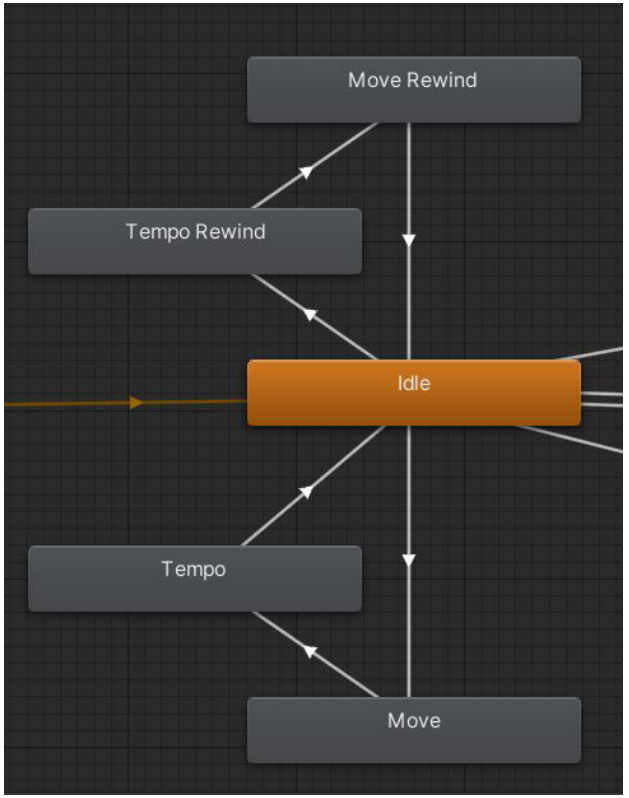
Le déplacement des TempoTiles se fait à l'aide d'un Lerp de la position de départ vers celle d'arrivée avec une valeur flottante appelée

```
public class TempoBehavior : StateMachineBehaviour
{
    variables

    Message Unity | 0 références
    public override void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)...

    Message Unity | 0 références
    public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)...

    2 références
    void NewTempoTile(Animator anim)...

    2 références
    void TempoTileIncr(Animator anim)...

    1 référence
    void tempoChange(Animator anim)...

    3 références
    bool LerpPos(GridTiles tile, bool flager, bool colorFlag, GridTiling g, string musicName)...

    3 références
    void UpdateAdjacentTileColonnes(GridTiles g, int x, int y, GridTiling gT)...
}
```

Rewind States

Si le joueur active le bouton de Rewind, il empreintera un autre cycle d'états de la StateMachine. Ces états contiennent les mêmes scripts de la classe StateBehavior, mais les méthodes et fonctions à l'intérieur ne réagissent pas de la même manière grace au paramètre booléen «Rewind».

Le mouvement est aussi activé avant les TempoTiles.

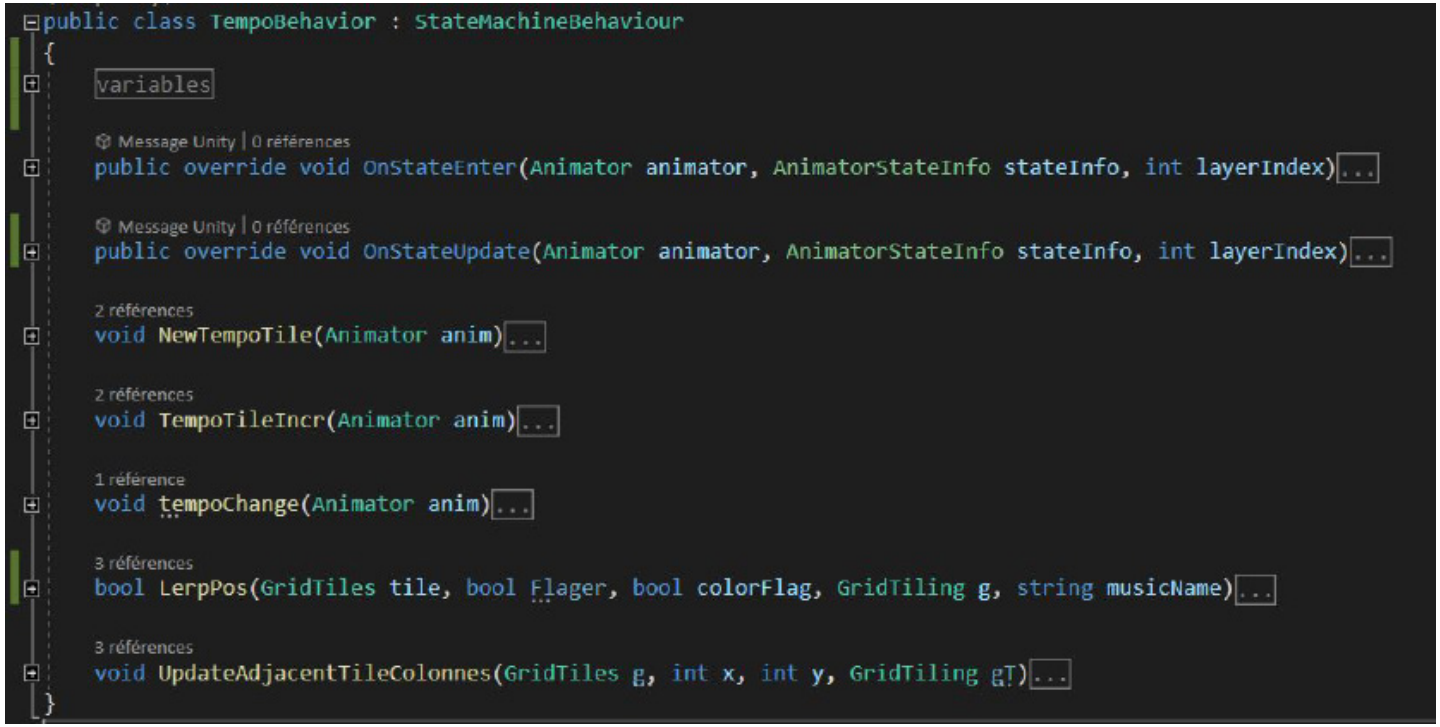


Enjeux majeurs

Pause Behavior

Le Script de «Pause Behavior» permet de lancer l'animation du menu pause et d'empêcher les autres états de s'activer.
Il utilise le script «DoCoroutine» pour faire le Lerp des Tiles de la scène.

Une fois la transition finie la state machine passe en état «Exit Pause» jusqu'à ce que le joueur quitte le menu pause dansquel cas l'animation de retour en jeu est activée puis la State Machine repasse dans l'état Idle pour laisser le joueur lancer des inputs.



Enjeux majeurs

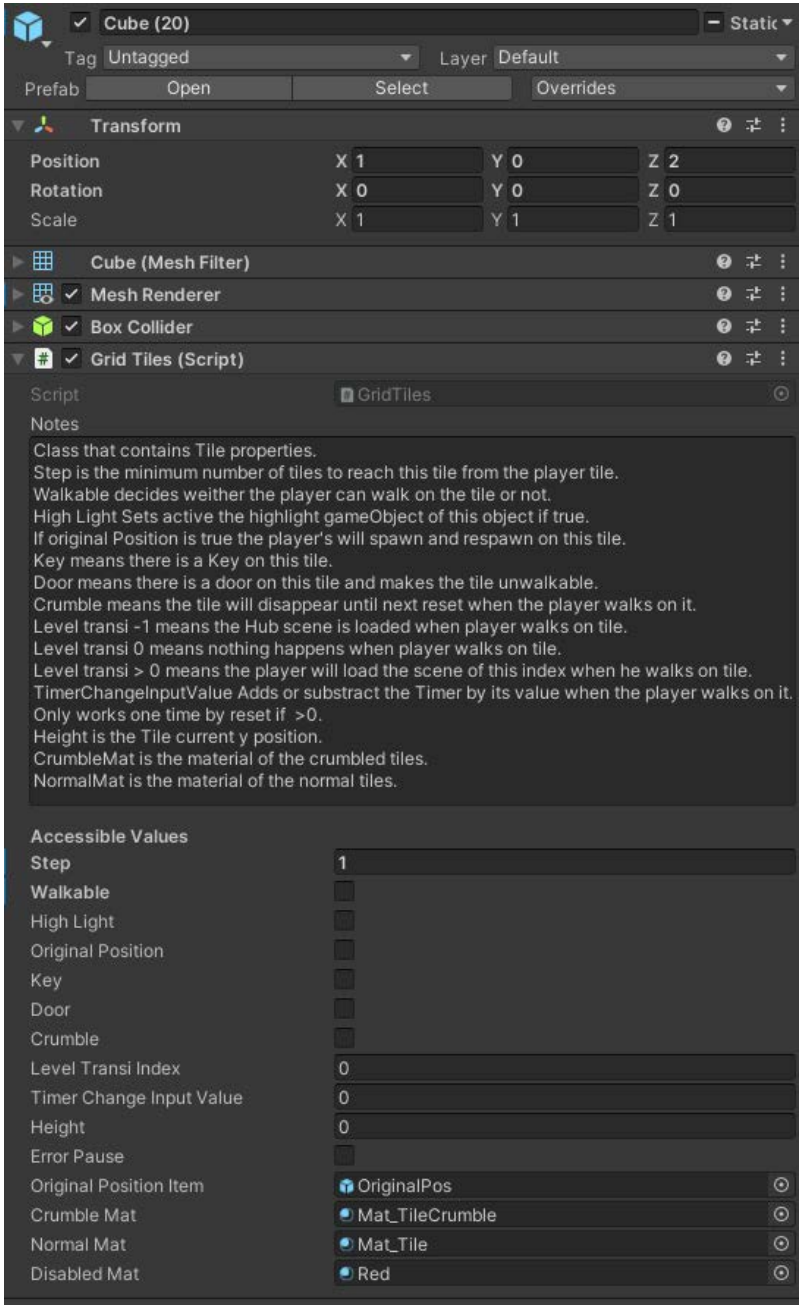
Level Editor

Un des enjeux du développement de notre jeu est de faire un éditeur de niveau efficace et intuitif pour faciliter un bon travail des level designers.

On Draw Gizmo

Afin de transformer directement dans la scène les objects en fonctions des actions des Level Designer, on utilise des fonctions «OnDrawGizmos()» ainsi que des valeurs booléennes à cocher / décocher sur les tiles. Ces booléens déterminent les effets spéciaux des tiles comme ‘crumble’ ou ‘key’.

Quand ces variables sont changées, leurs signes s’appliquent directement dans l’éditeur de niveau (instanciation des objets sur la tiles, changement de matériaux, etc.).



Enjeux majeurs

Fenêtre de Tiling

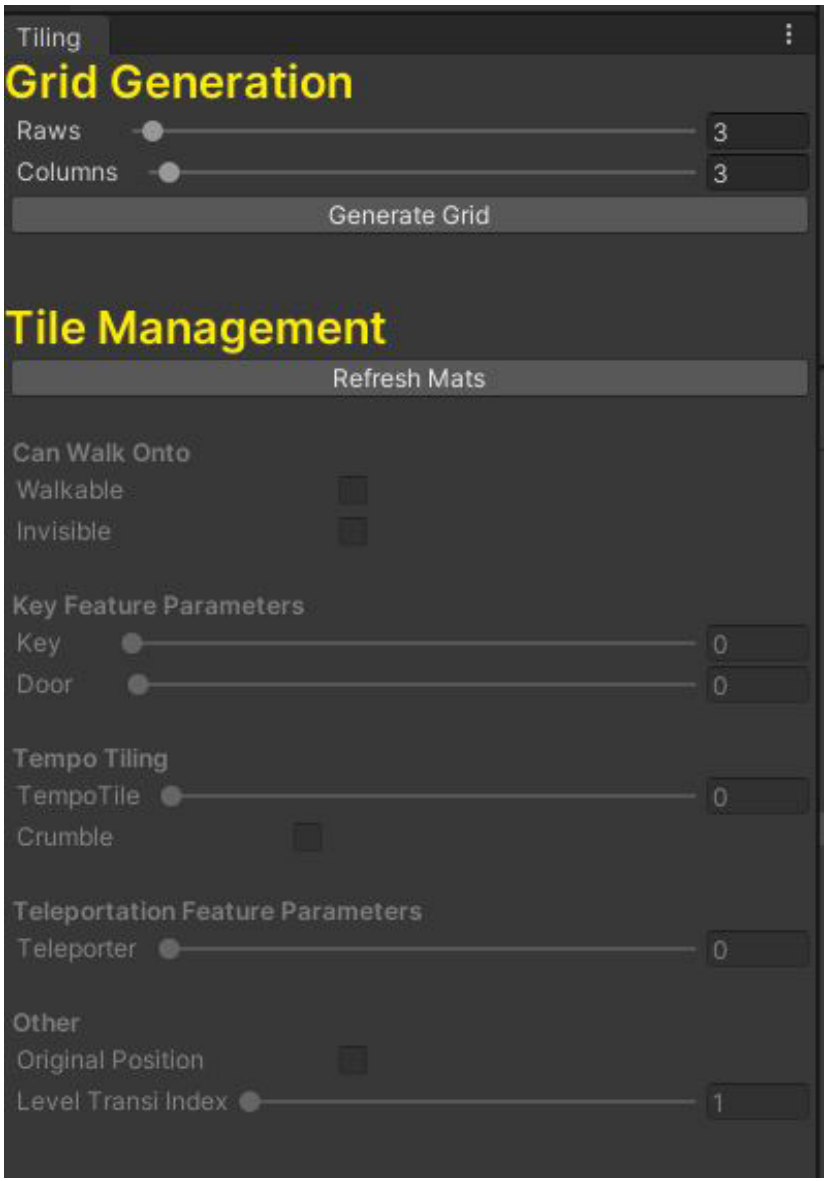
Le joueur à besoin d'accéder à différents objets souvent enfant de plusieurs parents pour changer ces valeurs dans les inspecteurs. Ce procédé fait perdre beaucoup de temps au level designer dans la conception de niveaux et rend le procédé moins optimisé.

Afin de faciliter la tache aux levels designer, nous avons créé une fenêtre d'édition de niveau appelé «Tiling». Cette fenêtre est divisée en 3 catégories :

- La génération de la grille, on trouve ici deux curseurs déterminant la taille de la grille qui doit etre généré, ainsi qu'un bouton permettant de générer la grille.

- Le raffraîchissement des matériels de la grille, ce bouton permet quand appuyé de raffréchir tous les matériels de la grille en fonction des valeurs rentrées par le joueur et de leur position les unes en fonction des autres.

- La gestion des tiles, cette catégorie contient toutes les variables permettant de changer les effets d'une tile. Quand ces variables sont changées, les tiles séléctionnées par le level designer sont transformées en conséquence. Quand aucune tile n'est séléctionnée, ces variables sont grisées et pas interactibles.



Enjeux majeurs

Pour créer cette fenêtre nous utilisons un script de la classe «EditorWindow».
Afin de générer la grille, le script change les varibles du script «GridGenerator» afin d'y activer la méthode public «generateGrid()».

Les matériaux sont actualisés en activant la fonction public «SetDirectionalMaterial()» et «SetTempoMaterial()» du script «GridTiling» de chaque tile de la scène.

Pour changer les variables des tiles séléctionnées, une variable local du script de l'éditeur est changé, elle est ensuite transférée à une variables transitoire présente sur chaque tiles séléctionnées par le level designer. Dans la frame qui suit, si la variable transitoire ne correspond pas à la variable de la tile, cette dernière est changé pour correspondre à la variable transitoire.

Les objects séléctionnés sont accessibles grâce à un foreach qui prend tous les objets contenant le script «GridTiles» présents dans un tableau contenant tous les objects séléctionnés par le joueur («Selection.gameObjects»).

Afin que les valeurs changées ne soient pas réinitialisées à la fin de la frame, nous utilisons la méthode «SetDirty()» qui permet d'enregistrer les valeurs dans la scène.

```
public class TileEditorWindow : EditorWindow
{
    variables

    [MenuItem("Window/Tiling")]
    0 références
    public static void ShowWindow()...

    Message Unity | 0 références
    private void Update()...

    Message Unity | 0 références
    private void OnGUI()...

    1 référence
    void GridGeneration()...

    1 référence
    void MatRefresh()...

    1 référence
    void AssignVariableValue()...

    1 référence
    void DrawVariablesInWindow()...

    1 référence
    void UpdateValues()...
}
#endif
```

Enjeux majeurs

Signes et Feedbacks

Un des enjeux majeur du projet était d'avoir une expérience affordante. Cet intention nécessitait des signes et feedbacks efficaces et précis.

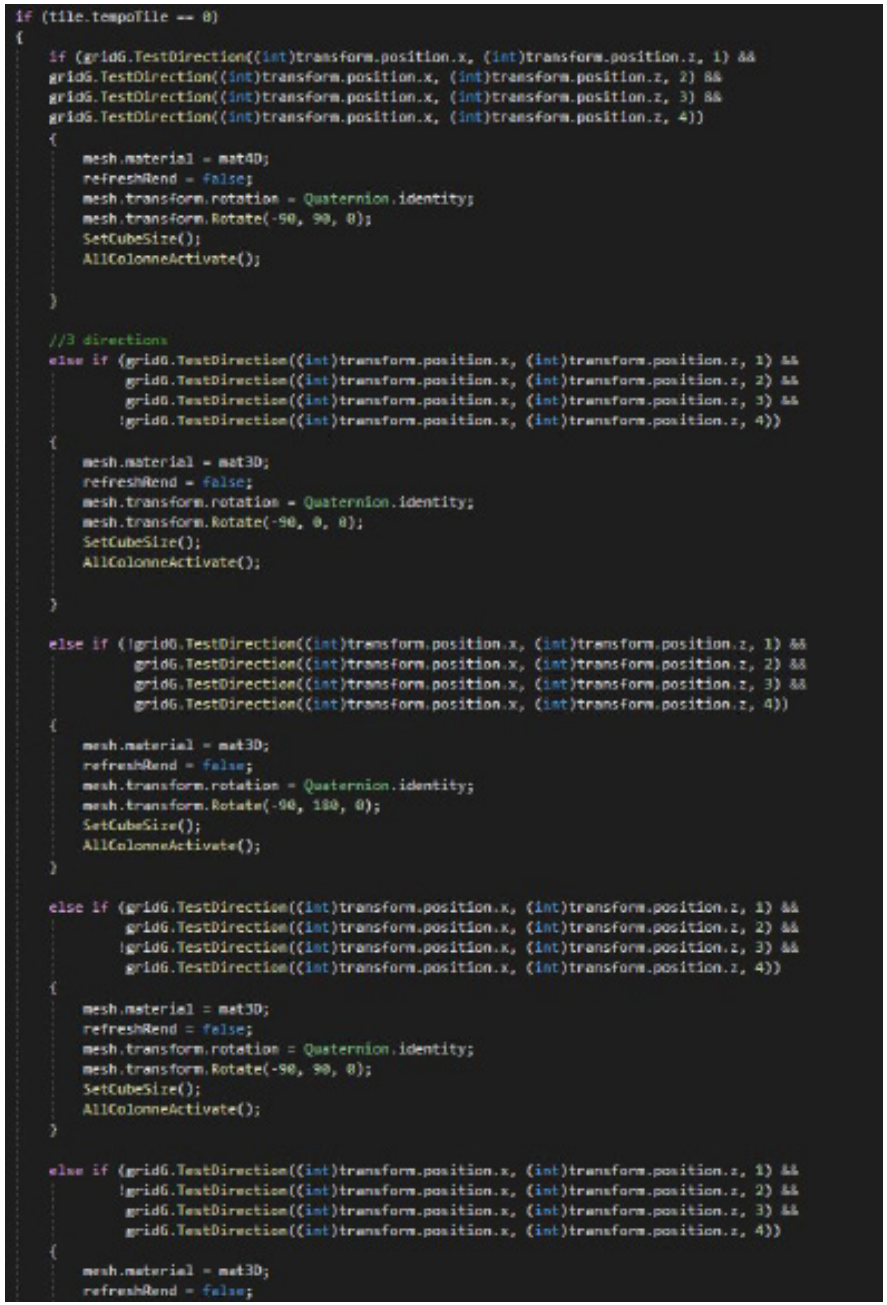
GridTiling - Accessibilité

Une des informations les plus importants de notre jeu était quelles tiles sont accessibles. Nous avons créé des matériaux se placant sur les tiles tracant des chemins entre les tiles ainsi que des lignes noirs délimitant les accès non employables par le joueur.

Ces matériaux ne pouvait pas être placé grace à des shaders à cause des conditions précise et parfois non prévisible qui les plaçaient.

Nous avons donc créé des méthodes de placement de matériaux procédural et automatisé afin d'avoir des signes efficaces. Ces méthodes s'appliquent régulièrement dans des circonstances précises afin de ne pas surcharger le jeu.

Elles tournent les tiles, changent leurs matériaux, active et désactive leurs colonnes et changent le scaling et la position des colonnes celon les nécessitées.



```
if (tile.tempoTile == 0)
{
    if (gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 1) &&
    gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 2) &&
    gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 3) &&
    gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 4))
    {
        mesh.material = mat4D;
        refreshRend = false;
        mesh.transform.rotation = Quaternion.identity;
        mesh.transform.Rotate(-90, 90, 0);
        SetCubeSize();
        AllColonneActivate();

    }

    //3 directions
    else if (gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 1) &&
            gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 2) &&
            gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 3) &&
           !gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 4))
    {
        mesh.material = mat3D;
        refreshRend = false;
        mesh.transform.rotation = Quaternion.identity;
        mesh.transform.Rotate(-90, 0, 0);
        SetCubeSize();
        AllColonneActivate();

    }

    else if (!gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 1) &&
             gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 2) &&
             gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 3) &&
             gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 4))
    {
        mesh.material = mat3D;
        refreshRend = false;
        mesh.transform.rotation = Quaternion.identity;
        mesh.transform.Rotate(-90, 180, 0);
        SetCubeSize();
        AllColonneActivate();
    }

    else if (gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 1) &&
            gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 2) &&
           !gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 3) &&
            gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 4))
    {
        mesh.material = mat3D;
        refreshRend = false;
        mesh.transform.rotation = Quaternion.identity;
        mesh.transform.Rotate(-90, 90, 0);
        SetCubeSize();
        AllColonneActivate();
    }

    else if (gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 1) &&
           !gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 2) &&
            gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 3) &&
            gridG.TestDirection((int)transform.position.x, (int)transform.position.z, 4))
    {
        mesh.material = mat3D;
        refreshRend = false;
```

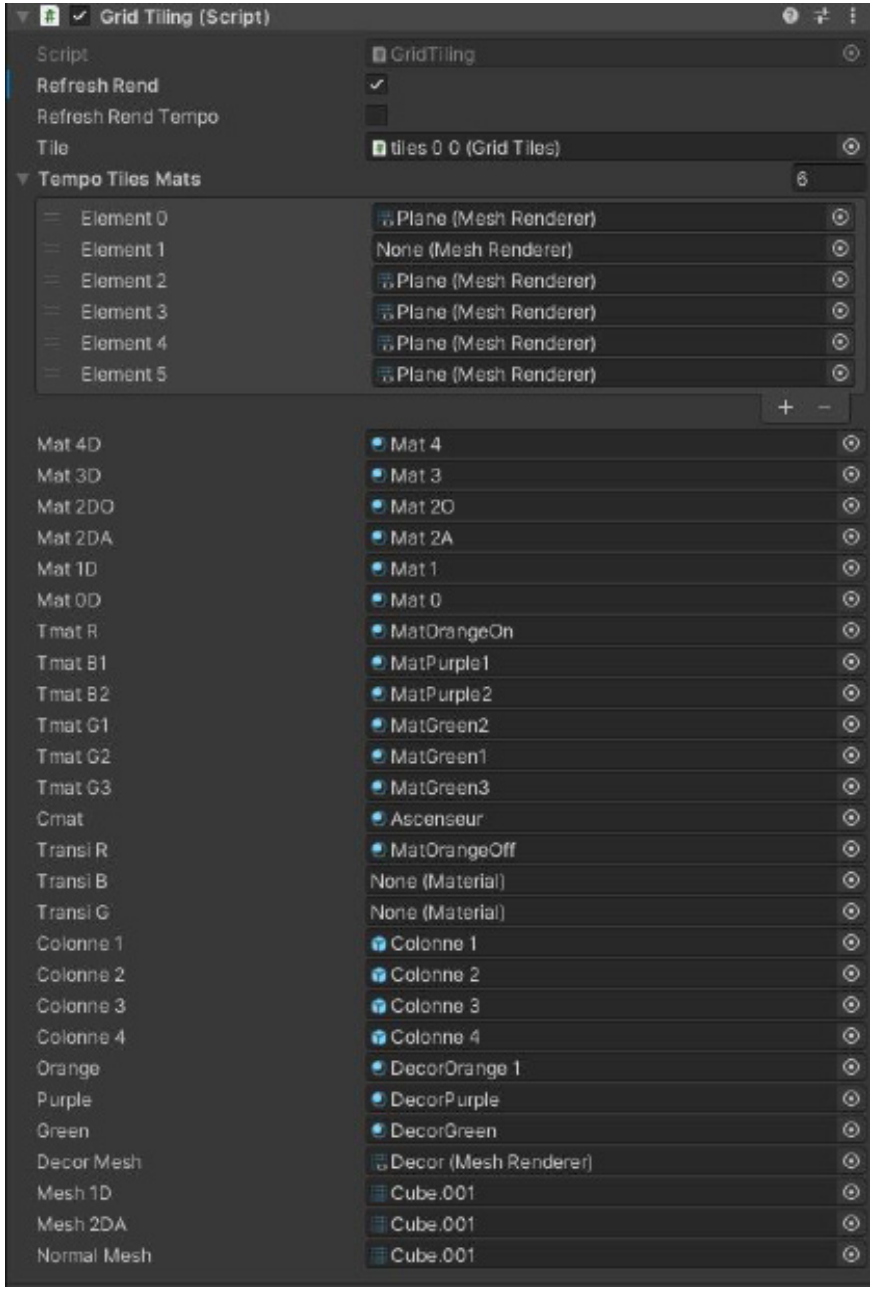
Enjeux majeurs

GridTiling - TempoTiles

Une autre information importante était le nombre de déplacements nécessaires avant qu'une TempoTile monte ou descende.

Ici aussi nous avons fait du placement de matériel procédural. Apres chaque déplacement nous changeons le matériel en fonction de valeurs inscrites sur la tile.

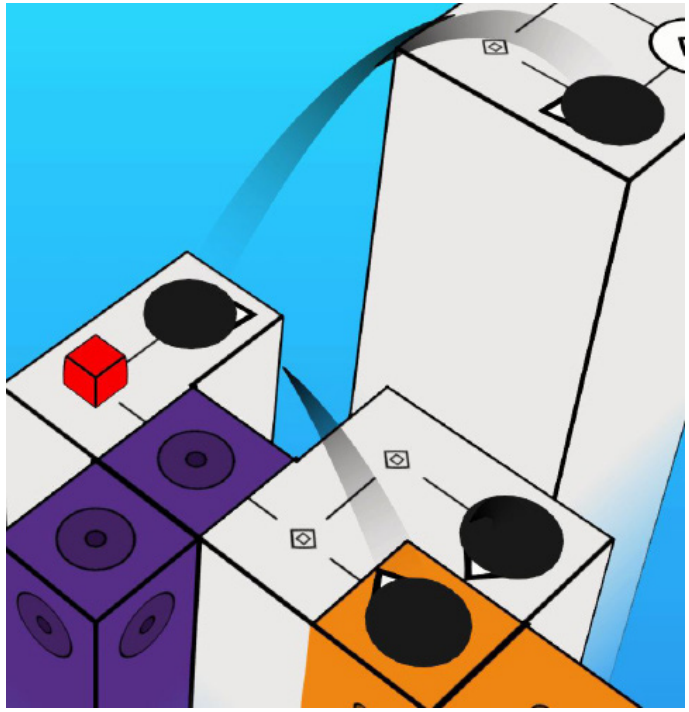
```
public void TempoTileMaterial()
{
    if (tile.tempoTile != 0 || tile.crumble)
    {
        if (tile.crumble)
        {
            mesh.transform.rotation = Quaternion.identity;
            mesh.transform.Rotate(-90, 0, 0);
            SetDirectionalMaterial();
            foreach (MeshRenderer m in tempoTilesMats)
            {

                //mesh.transform.Rotate(-90, 0, 0);
                //m.material = Cmat;
                //m.material.Lerp(m.material,Cmat,Time.deltaTime);

            }
              grid0.TestDirection((int)transform.position.x, (int)transform.position.z, 1)
            grid0.TestDirection((int)transform.position.x, (int)transform.position.z, 2);
            grid0.TestDirection((int)transform.position.x, (int)transform.position.z, 3);
            grid0.TestDirection((int)transform.position.x, (int)transform.position.z, 4);
            SetCubeSize();
            AllColonneActivate();*/
            refreshRendTempo = false;
        }

        if (tile.tempoTile == 1)
        {
            mesh.transform.rotation = Quaternion.identity;
            mesh.transform.Rotate(-90, 0, 0);
            mesh.material = TmatR;
            //Material curM = mesh.material;
            //MaterialLerping(TmatR, TmatR);
            // mesh.material.Lerp(mesh.material,TmatR,Time.deltaTime*2);
```



Enjeux majeurs

TilingEditor - Teleporter

Les téléporteurs ont un fonctionnement particulier, ils envoient forcément le joueur vers un autre TP mais ne sont pas forcément des récépteurs de téléportation. Il fallait donc qu'il y ait des signes efficaces indiquant au joueur ou il apparaîtra si il rentre dans un téléporteur.



Le premier signe que nous avons utilisé est un triangle sortant du téléporteur et pointant dans la direction de son récépteur. Ce signe est efficace mais donne une position approximative du récépteur.

Nous avons ajouté un second signe complétant le premier. De manière ponctuelle une sphère est instancié sur chaque TP se déplace jusqu'à sa destination en suivant une courbe. La sphère n'a pas de renderer et a une trail noir formant un pont entre le tp et sa destination.

Ce signe ne suffira pas non plus vu qu'il n'est pas tout le temps activé et transparent le rendant plus précis mais moins visible.

Afin d'avoir la courbe que suit le sphère, deux coordonnées de vecteurs font des Vector3.Lerp() entre trois points (le téléporteur, son récépteur et un point médian placé entre le téléporteur et son récépteur en hauteur). La sphère fait un lerp entre les 2 coordinées de vecteur effectuant des lerp.
A la fin du lerp de la sphère, elle est détruite et la variable flottante d'interpolation est réinitiallisée à 0, après un timer la sphère est réinstanciée.

```
p1 = new Vector3(transform.position.x, transform.position.y + 0.6f, transform.position.z);
p2 = new Vector3(tpTarget.x,tpTarget.y +0.1f, tpTarget.z);
p3 = (p1 + p2) / 2 + new Vector3(0, Mathf.Abs(p1.y-p2.y) + 1, 0);

interpolateAmount = (interpolateAmount + Time.deltaTime) % 1;

p1p3 = Vector3.Lerp(p1, p3, interpolateAmount);
p3p2 = Vector3.Lerp(p3, p2, interpolateAmount);
p1p2 = Vector3.Lerp(p1p3, p3p2, interpolateAmount);

if (Trail != null)
    Trail.position = p1p2;
```

Direction Artistique Visuelle

- Intentions -
- Recherches artistiques -
- Références ludiques -
- Réalisation des assets -

Intentions

Notre jeu est un puzzle gamme 3D en vue isométrique. L'avatar du joueur se déplace sur un espace quadrillé et son objectif est d'atteindre la case de fin de niveau. Le jeu est construit autour d'un «bloc» de Level Design en particulier, qui est la TempoTile : la case rythmée en quelque sorte.
Les TempoTiles montent et descendent en fonction du nombre de déplacements de l'avatar (voir schéma ci-contre), en faisant se mouvoir ces TempoTiles, le joueur se crée de nouvelles opportunités et donc de nouveaux chemins.
Le principal challenge des puzzles est donc lié à la difficulté de trouver le bon chemin pour atteindre la case de fin de niveau.

Notre première intention est que la Direction Artistique desserve toujours la clarté des niveaux. Cela semble être logique de ne pas juste habiller notre jeu avec un thème. Mais dans un puzzle gamme comme le nôtre, cela est un enjeu essentiel. Les niveaux requièrent la réflexion, la patience et la logique du joueur, si nous surchargeons le décor d'asset 3D et autres, cela risque de nuire à la lecture. Les mécaniques dans les puzzles se doivent d'être claires et la DA doit soutenir ce principe.

Par conséquent, nous avons adopté une direction artistique plus «casual», donc pas de géométries complexes et un minimum d'assets 3D. Dans cette même intention de lisibilité, nous n'avons pas rajouté de décor alentour à l'instar de *Lara Croft GO* et *Hitman GO*.

La deuxième intention est d'avoir une identité forte, un style propre à notre jeu. Il ne s'agit pas d'avoir des modèles complexes et une multitude d'assets, mais au contraire avoir une direction artistique particulière se départageant des normes visuelles des jeux «casual» ou «hyper-casual» à l'aide d'une mise en scène et de shaders particuliers.

Nous aimerions donc nous orienter sur un travail conséquent sur la colorimétrie: avoir une palette de couleur cohérente, nuancée ... également se pencher sur la question du daltonisme chez certains joueurs. Mais aussi un travail sur le lightening de la scène, la skybox (c'est-à-dire le fond de notre scène). Une recherche sur différents styles tels que le Toon / Cell Shading, le bloom du lightening, l'ambiant occlusion de la scène ou encore un aspect peinture, etc.

Recherches Artistiques

Le rêve et le nuage

Continuation Of The Dream, Artem Chebokha

Sasha's Dream, Artem Chebokha

Nature Salvation, Artem Chebokha

Lost And Forgotten, Artem Chebokha

Recherches Artistiques

Le rêve cartoon

Le Petit Prince, Antoine de St Exupery

Logo de DreamWorks Animation

Le Petit Prince, Antoine de St Exupery

Aplats de couleurs et formes simples

Gris, Devolver Digitals

Changement de couleurs et de matières

Dyptique Marilyn, Andy Warhol

Camaieu de pot de peinture

Recherches Artistiques

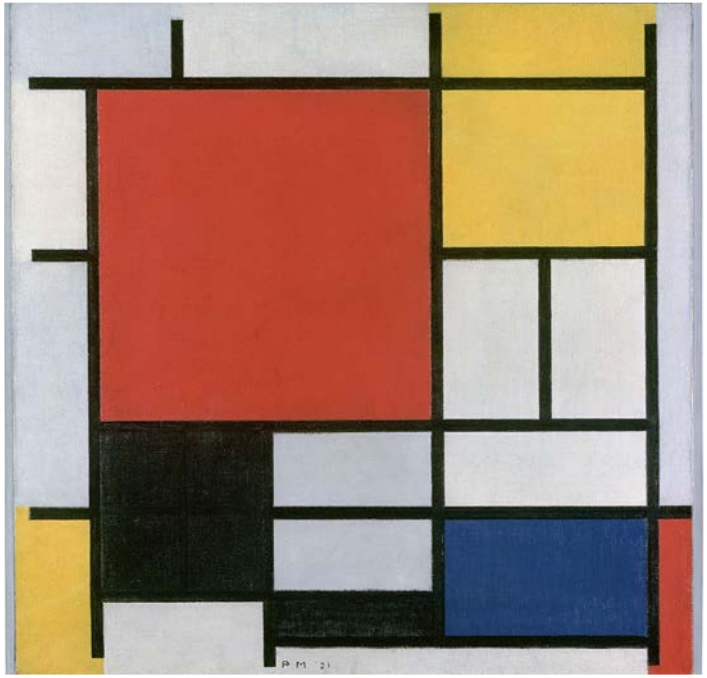
Robert Morris

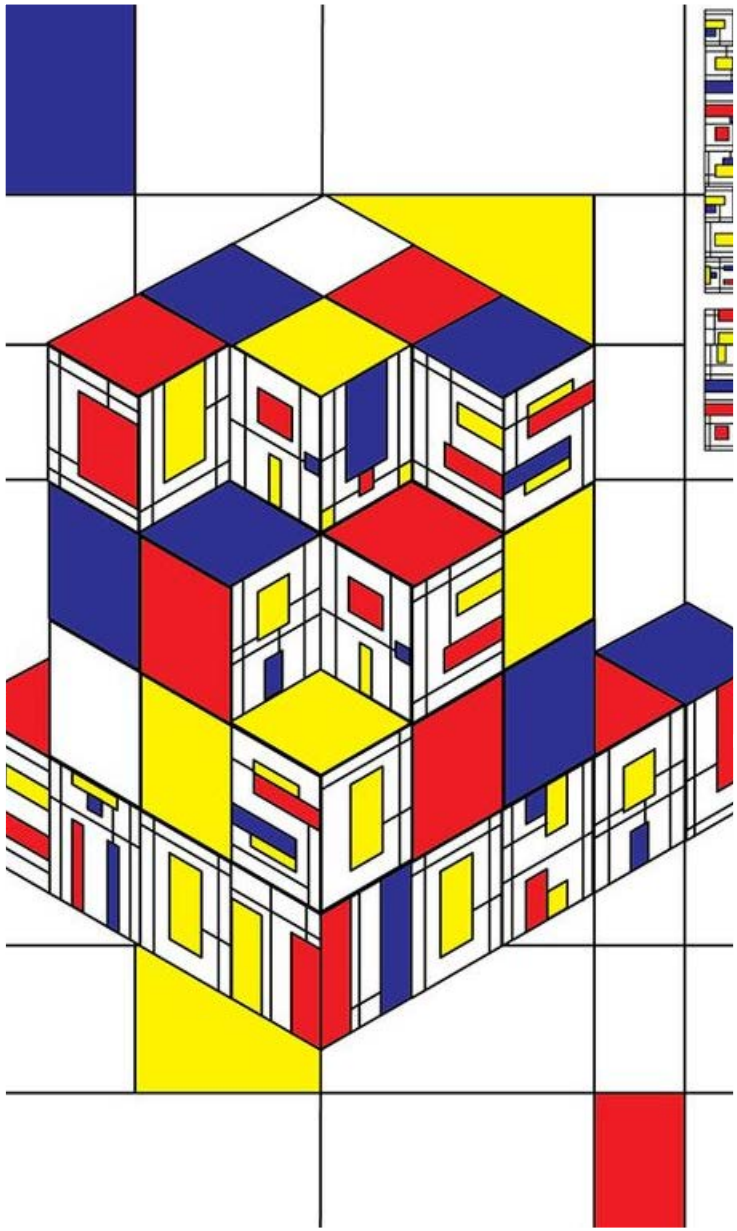
Robert Morris est un artiste plasticien, scénographe. Il est célèbre pour ses installations mettant en scène des objets aux formes géométriques primaires et où le rapport à la lumière est prédominant.
On peut ressentir son influence dans notre travail notamment avec l'utilisation d'à-plats de couleurs, la géométrie primaire et en particulier les cubes. L'intention minimaliste et très épurée dans le travail de Robert Morris nous a aussi suivis tout au long du projet.

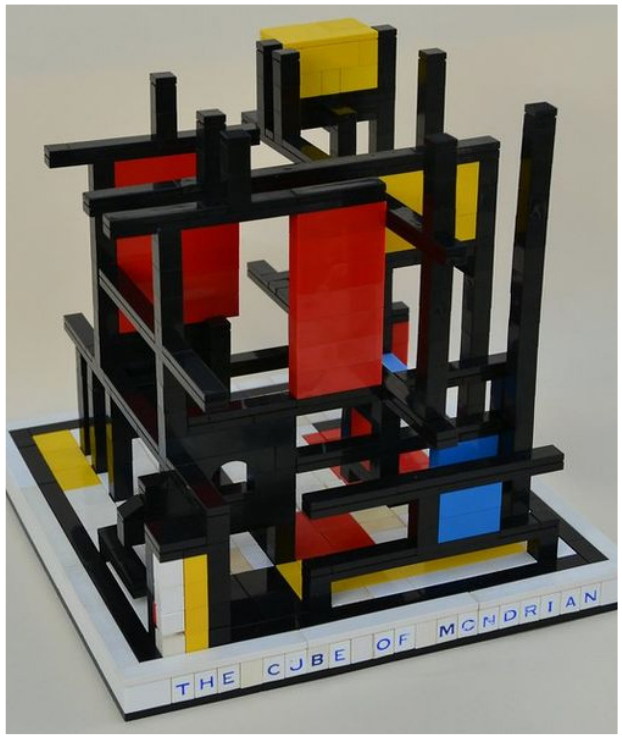
Recherches Artistiques

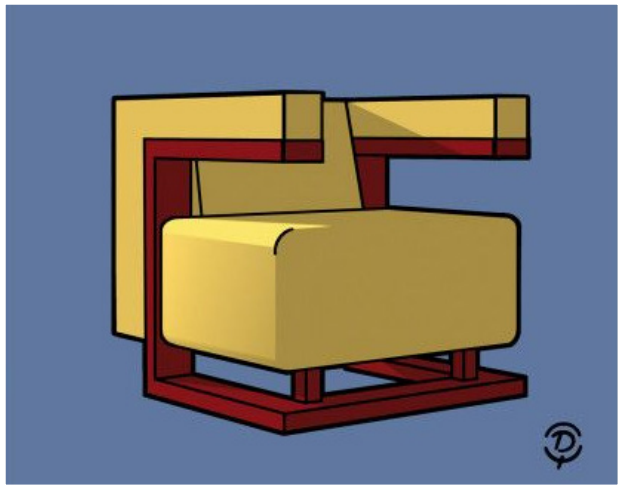
De Stilj

Le mouvement De Stilj, traduit littéralement par «Le Style», est un mouvement artistique né aux Pays-Bas au début du 20e siècle. Il est caractérisé par des compositions trichromatiques et l'omniprésence des contours noirs.
Le travail des néoplasticiens et des architectes de ce mouvement en particulier Piet Mondrian peut se ressentir dans Araka.
Tour d'abord avec l'utilisation de trois couleurs pré dominantes qui sont chez nous l'orange, le vert et le violet, puis l'utilisation de la «Outline» noire pour intensifier le relief et créer de la profondeur. Mais aussi tout comme avec Robert Morris l'utilisation de formes géométriques simples.



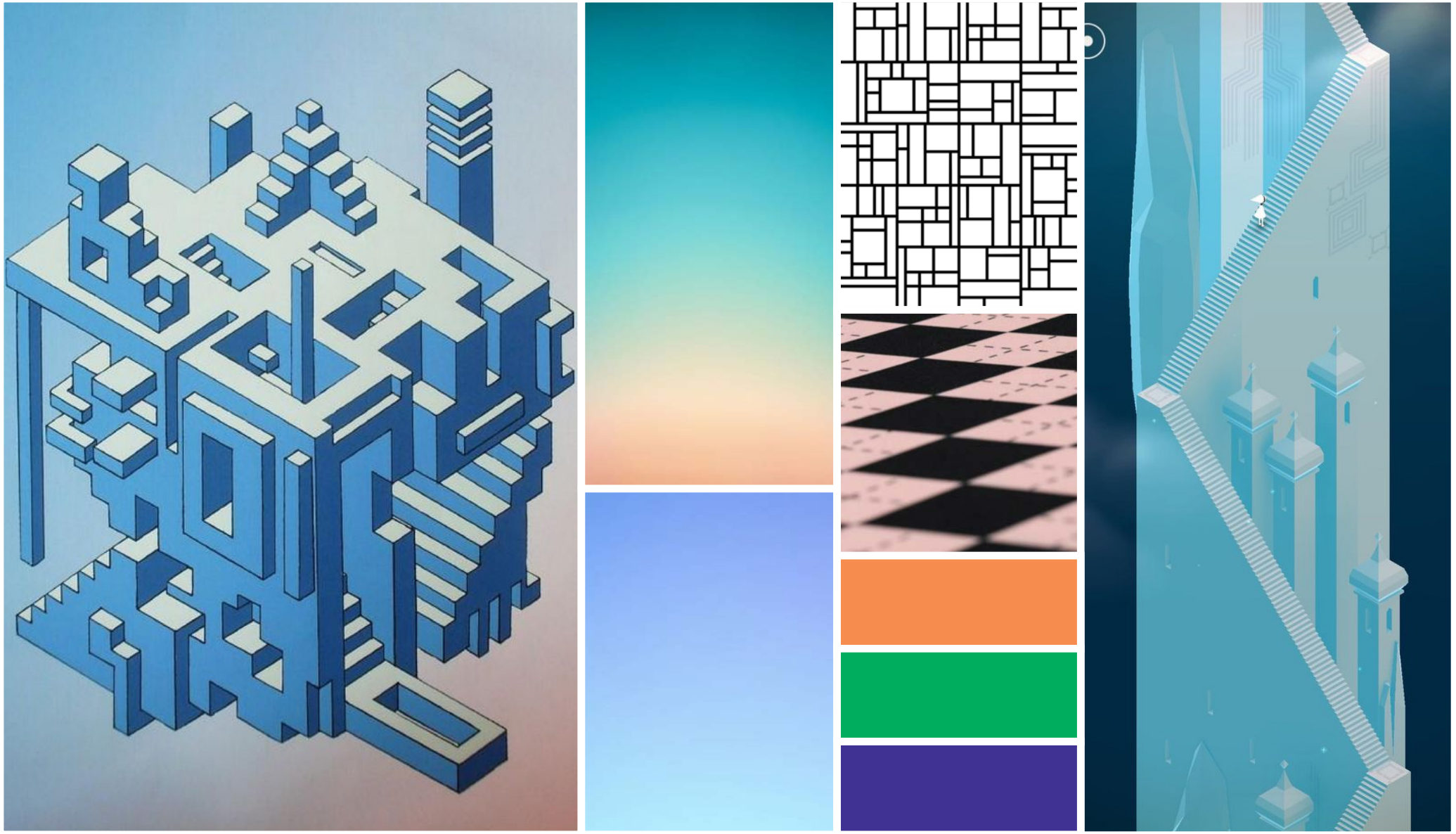






Recherches Artistiques

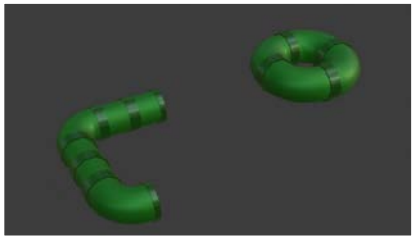
MoodBoard

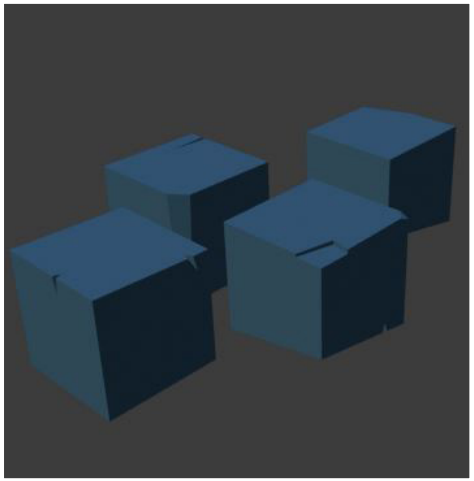


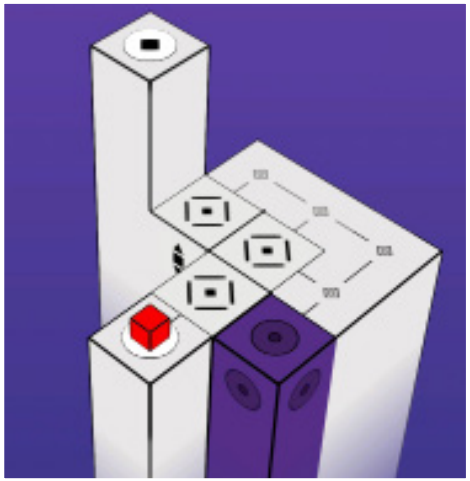
Recherches Artistiques -Décor

Recherches

Nous avons eu un bon nombre de reboot sur le projet en termes de décor et d'asset pour les tiles en particulier. Et donc avant qu'on ait ce qu'on a aujourd'hui nous sommes passés par plusieurs passes. Les plus marquantes ont été les égouts avec de la tuyauterie à la place des cases, ou encore un tiling plus rocailleux avec une couleur bleue nuit. Le problème récure a tous ces tests c'est qui rendaient les niveaux beaucoup moins lisible et les challenges moins clairs.

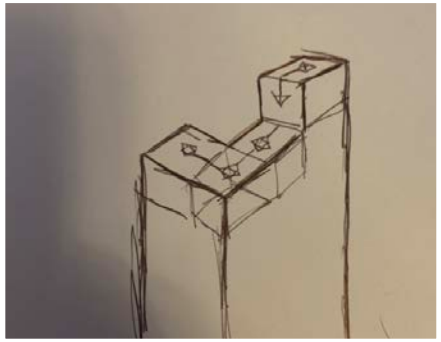


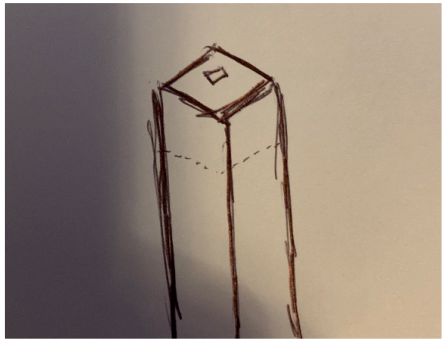


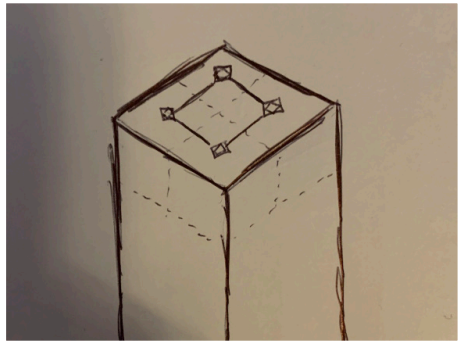


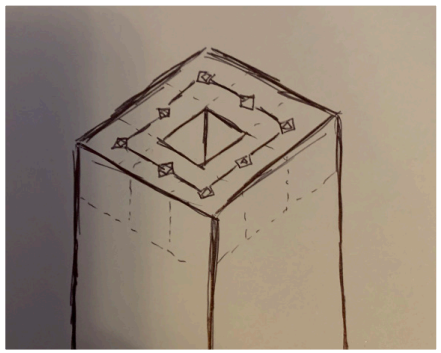
Sketches

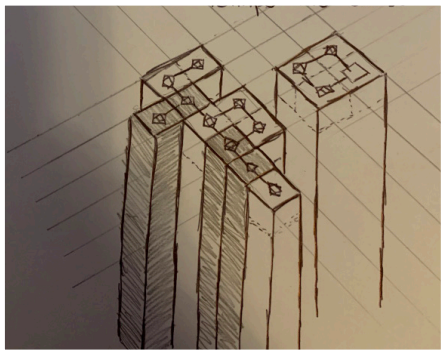
Ce sont les derniers schémas qui ont permis la visualisation de la direction artistique du projet. Le choix d'avoir étiré les plateformes vers le bas de l'écran est en grande partie esthétique, mais il rajoute en plus de cela de la profondeur à l'écran, ce qui est pratique pour venir dynamiser des niveaux qui ne semblent parfois très vides et flottants pour aucune raison.
La signalétique au sol et le quadrillage sont réfléchis de manière à ce que le joueur sache toujours là où il peut ou non se déplacer. Les textures sur les blocs sont dynamiques, c'est-à-dire qu'elles changent en fonction des possibilités de déplacement.

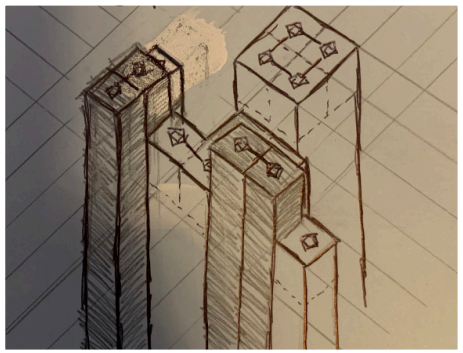










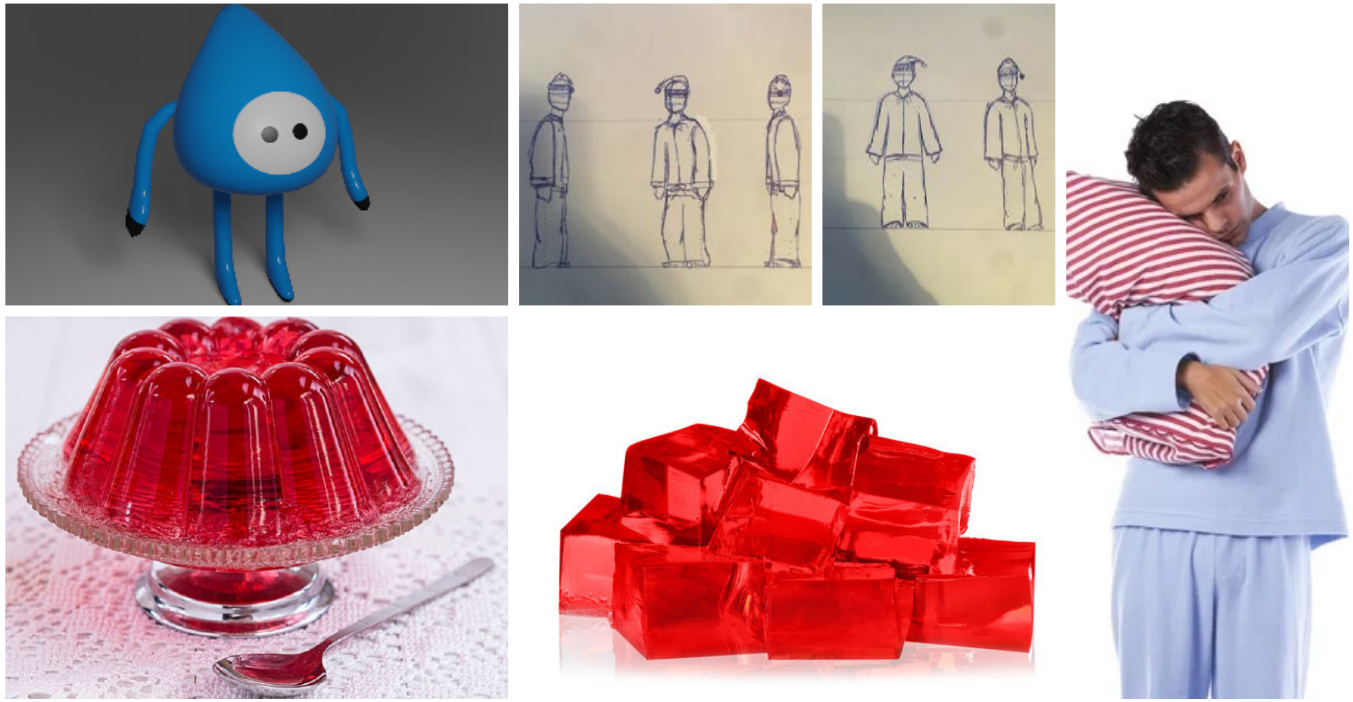


Recherches Artistiques - Avatar

Recherches

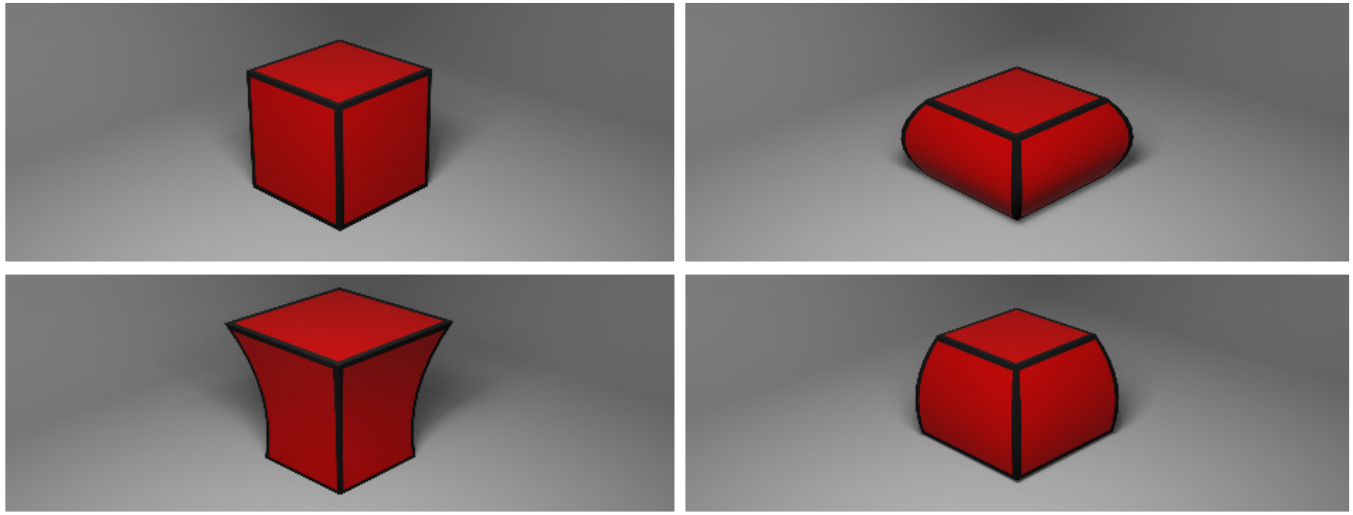
Notre première idée était d'avoir comme avatar, un jeune adulte en pyjama à peine réveillé qui veut juste retrouvé son lit pour se rendormir. L'idée était intéressante surtout du point de vue du déplacement, car l'amplitude de son pyjama pouvait offrir une fluidité et une légèreté aux mouvements. Mais ce choix et d'autres essais comme celui de l'avatar «Goutte d'eau», un choix un peu plus onirique, créaient un contraste dérangeant avec la simplicité de nos niveaux.

On a alors cherché dans nos références et nous nous sommes penchés vers quelque chose de plus gélatineux comme dans *Reky* ou *Vectronom*. La jelly anglais et son comportement ont été assez inspirants pour comprendre la physique de certains mouvements.



Resultat Final

Notre avatar actuel est donc un cube a la couleur rouge vif, il est impacté par ce fameux contour noir, mais son déplacement et sa physique gélatineuse contraste avec la rigidité du décor et donc plus facile a remarqué.



Références ludiques

Reky est une de nos plus grandes inspiration, tant sur sur l'aspect visuel que sur la manière de nous projeté pour notre direction artistique finale. En effet on constate quelques similitudes entre les deux jeux. Tout d'abord l'omniprésence de la outline qui sert à bien apercevoir les reliefs et comprendre la perspective des puzzles, mais aussi l'utilisation d'une palette de couleur simple avec pour chaques blocs de level design une couleur qui lui est propre. Nous n'avons pas dutout la même manière de produire nos asset et Reky offre un game feel et des types de puzzles completement différent des nôtres.

Golf Peaks quant à lui est inspirant sur la manière qu'il a d'aborder la progression. Le style de caméra et de grille de notre jeu y fait référence. Mais ce qui nous intéresse c'est comment est gérée la difficulté des niveaux dans l'ordre, comment est implémentée une nouvelle mécanique / bloc LD, et comment est fait le tutoriel.
Outre tous ces aspects, Golf Peaks possède un univers graphique qui lui est propre et il se rapproche de ce que l'on aimerait faire : minimalisme et simplicité, efficacité et identité forte.

Références ludiques

Lara Croft GO et Hitman GO sont deux références qui nous servent beaucoup sur le Game Design et le Level Design, mais aussi beaucoup sur l'UX et plus précisément sur l'UI, notamment le tiling des cases au sol ou encore comment fait-on comprendre au joueur qu'il peut se déplacer sur telle case et pas tel autre. Graphiquement, ce sont des jeux à licence, le public concerné par ces jeux s'attend à retrouver la patte graphique ou l'univers visuel de leur jeu. Et comme précédemment expliqués, nous aimerions partir sur quelque chose d'assez efficace et simple (ce qui relève d'une grande complexité).

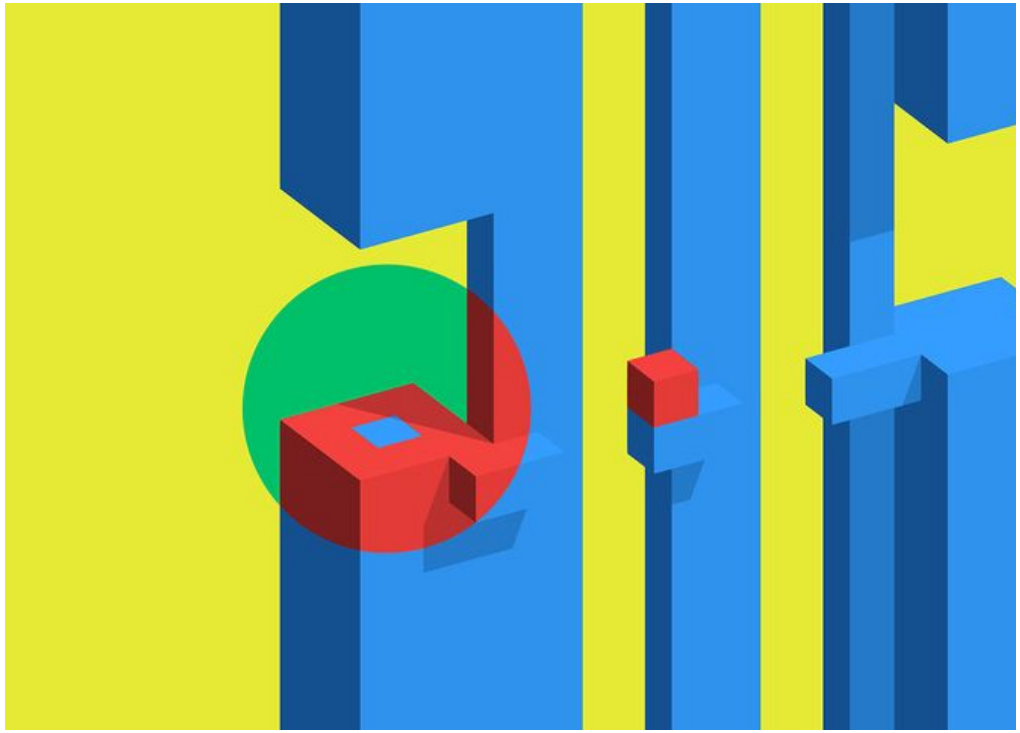
Monument Valley est une de nos premières références. Tant sur l'aspect Game Design que sur le visuel et le Game feeling. Son aspect épuré sur tous les points de vue le rend fascinant et la direction artistique du jeu est très inspirante sur ce que l'on veut produire. Autant sur le Sound Design et la composition sonore avec de la réverbération, du pitch et des assets sonore très spatialisé, que sur le visuel avec une identité puissante qui s'inspire grandement du travail d'Escher (un architecte), mais qui reste propre à l'ADN du jeu. Sa géométrie paraissant simple, mais étant en réalité complexe, ses à-plats et ses nuances de couleurs se rapprochent de cette identité graphique que l'on cherche à créer.
Le travail sur l'avatar du personnage retient aussi beaucoup notre attention, tout comme celui de Journey, il n'est pas réellement identifiable, mais le joueur peut facilement l'incarner. Son déplacement est une composante essentielle au Game feeling du jeu et à la juicyness de l'expérience.

Références ludiques

Vectronom tout comme Reky est une de nos grandes inspirations. Il est aussi influant sur le Game Design de notre jeu qu'il est une référence d'un point de vue visuel.

Le point sur lequel il nous a le plus inspiré et sur lequel on peut ressentir son influence, c'est sur l'avatar et tout particulièrement sur son déplacement. Nous voulions tout comme dans Vectronom avoir un avatar un peu gellatineux au déplacement rebondi et qui ogffre une éxperience assez amusante, chaques déplacement soit différent de l'ancien et que le iddle et l'attérissage d'un déplacement fasse vraiment ressentir ce côté bouncy. Cet apparence et ces metrics font en sorte que l'avatar ressorte un peu plus du décor comme si ce cube contrastait avec la rigidité des autres cubes qui composent le décor.

On pourrait aussi nous comparer sur l'utilisation d'à-plats de couleurs et sur la simplicité des niveaux du point de vue visuel.



Direction Artistique Sonore

- Intentions -
- Recherches sonores & musicales -
- Intégration FMOD -
- Tableau de sons -

Intentions

Première piste

L'avatar que l'on contrôle, peu importe sa forme, est seul dans les niveaux et ne sera pas amené à rencontrer d'autres formes de vie. La première intention que nous voulions exploiter est la retraite du monde et l'isolement de l'avatar. Des ambiances très spatiales, des sons avec une certaine amplitude seraient cohérents et rentreraient bien dans ce début d'univers que nous aimerions exploiter.

Quelques idées

Notre jeu étant un jeu de puzzle avec une forte apparence casual, nous voulions des sons d'actions et d'UI très peu présent pour laisser la majeure partie de l'information sonore aux TempoTiles. Nous ne voulions également pas partir dans la spatialisation des sons puisqu'il n'y avait pas d'intérêt en termes de gameplay et de mécaniques d'en avoir, nous avons donc opté pour une approche purement 2D pour l'intégralité des sons du jeu.

Intentions

Les TempoTiles

L'avatar du joueur se déplace sur un espace quadrillé et son objectif est d'atteindre la case de fin de niveau et notre jeu est construit autour d'un ingrédient de Level Design particulier, qui est la tempo tile. La case rythmée en quelque sorte.
Les Tempo tiles constituent la core mécanique du jeu, elles montent et descendent en fonction du nombre de déplacements de l'avatar, en faisant se mouvoir ces Tempo tiles le joueur se créer de nouvelles opportunités et donc de nouveaux chemins.

Le principal challenge des puzzles est donc lié à la difficulté de trouver le bon chemin pour atteindre la case de fin de niveau.Il est donc primordial dans notre choix de design sonore que le joueur puisse savoir lorsqu'une certaine Tempo tile est levée et lorsqu'une autre ne l'est pas.

Pour ce faire, nous avons réalisé une première composition sur Beepbox constitué d'une mélodie principale auquel on ajoute 3 pistes d'instrument : le kick, les cuivres et les cordes. Nous avons ensuite associé chacun de ces instruments secondaires à une famille de TempoTile distincte (Exemple: lorsque la Tempo tile bleu est activée la piste des cuivres s'ajoute à la mélodie principale).

Les TempoTiles

Par conséquent lorsqu'une tile revient à sa position initiale, c'est-à-dire lorsqu'elle est baissée, la piste qui lui est affiliée se désactive.

Cela crée un effet d'orchestre, de musique vivante au flow des mouvements du joueur, qui même avec notre premier placeholder était très agréable à l'oreille et très intéressant à exploiter.

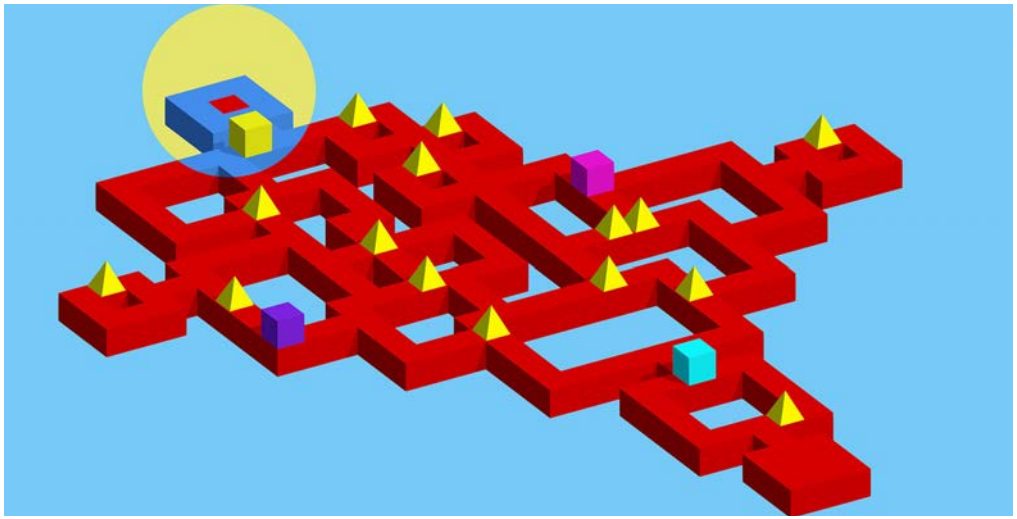
De plus ce choix de design renforce le côté émergent der notre jeu, puisqu'après plusieurs playtests nous remarquons que les joueurs font souvent le choix de jouer avec le rythme et par conséquent être dans les temps. Cela rajoute une dimension musicale au jeu.

Recherches sonores et musicales

Monument Valley possède des sonorités mystérieuses et oniriques qui sont deux caractéristiques que nous aimerions exploitées, c'est donc une première piste d'inspiration pour certains niveaux.

Vectronom nous inspire tout d'abord puisque c'est un puzzle gamme en isométrique, mais tout d'abord puisque la DA et la charte sonore sont très bien réalisé et, car elles découlent du meaningful play.

Journey est une référence pour nous sur l'aspect ruines / civilisation ancienne, mais aussi sur l'isolement et la retraite du monde que le joueur est amené à ressentir.



Référence de la core feature sonore

Nous avons un jeu dont la mélodie est influencée par la position des TempoTiles et donc directement influencée par les déplacements du joueur. Cela nous a été inspiré en partie par une phase de gameplay très connu de Castlevania : Lords of Shadow, le niveau de la boite à musique.

Dans ce niveau, le joueur est piégé dans une boite à musique et doit récupérer des cylindres de mélodie pour progresser, en changeant la disposition du niveau, ce qui va également altérer la mélodie de la boite à musique.

Le jeu étant plongée dans une ambiance très sombre et sérieuse, avoir un niveau dont la mélodie est celle d'une douce boite à musique donne un aspect très onirique et spécial à ce niveau, qui est aussi une source d'inspiration intéressante pour nous.

La mécanique d'avoir un level design changeant selon une mélodie et que ce soit le joueur qui décide de comment la modifier nous à interpelé et nous a permis d'avancer bien plus précisément vis-à-vis de nos choix de design sonore.

Intégration FMOD

Sons ponctuels

L'intégration des différents sons ponctuels FMOD c'est fait en avec des *FMODUnity.RunTimeManager.PlayOneShot(event:/)*.

Musique et Instruments de Tempo Tiles

Pour les sons de musique thème et d'instruments joués en fonction des TempoTiles activées, je crée en début au début de la scène une instance de chaque piste de musique et la joue.

En fonction des TempoTiles activées je réduis le volume des instances d'instrument à 0 ou le réaugmente à 1.

Au reload de la scène ou aux pops de l'écran de fin de niveau de stop toutes les instances avec des FadeOuts et release les instances pour récupérer la mémoire allouée.

Améliorations futures

Nous allons essayer de changer la musique de la Tempo Tile avec un tempo de 1 par un kick s'activant ponctuellement. Au lieu de faire un changement instantané du volume des instruments, nous allons faire une interpolation progressive (Lerp) qui sera moins abrupte.

Nous ne stopperons et relâcherons plus les instances de musique et d'instruments au reload de scène et à la transition de scène, mais réduirons le volume à 0 pour ne pas avoir de changement brusque de la musique en la reloadant et pour éviter une opération de réallocation de la mémoire en recréant une instance.

Pour les futures améliorations, nous allons essayer de changer la musique de la Tempo Tile avec un tempo de 1 par un kick s'activant ponctuellement.

Au lieu de faire un changement instantané du volume des instruments nous allons faire une interpolation progressive de la valeur de volume (Lerp) afin de smooth les transitions.

Au lieu de Stop et Release les musiques instruments nous mettrons leurs volumes a 0 et les réactiveront pour enlever la césure créée par le rebot du niveau et pour retirer l'opération de réallocation de mémoire lorsqu'on recrée une instance à chaque scène.

Intégration FMOD

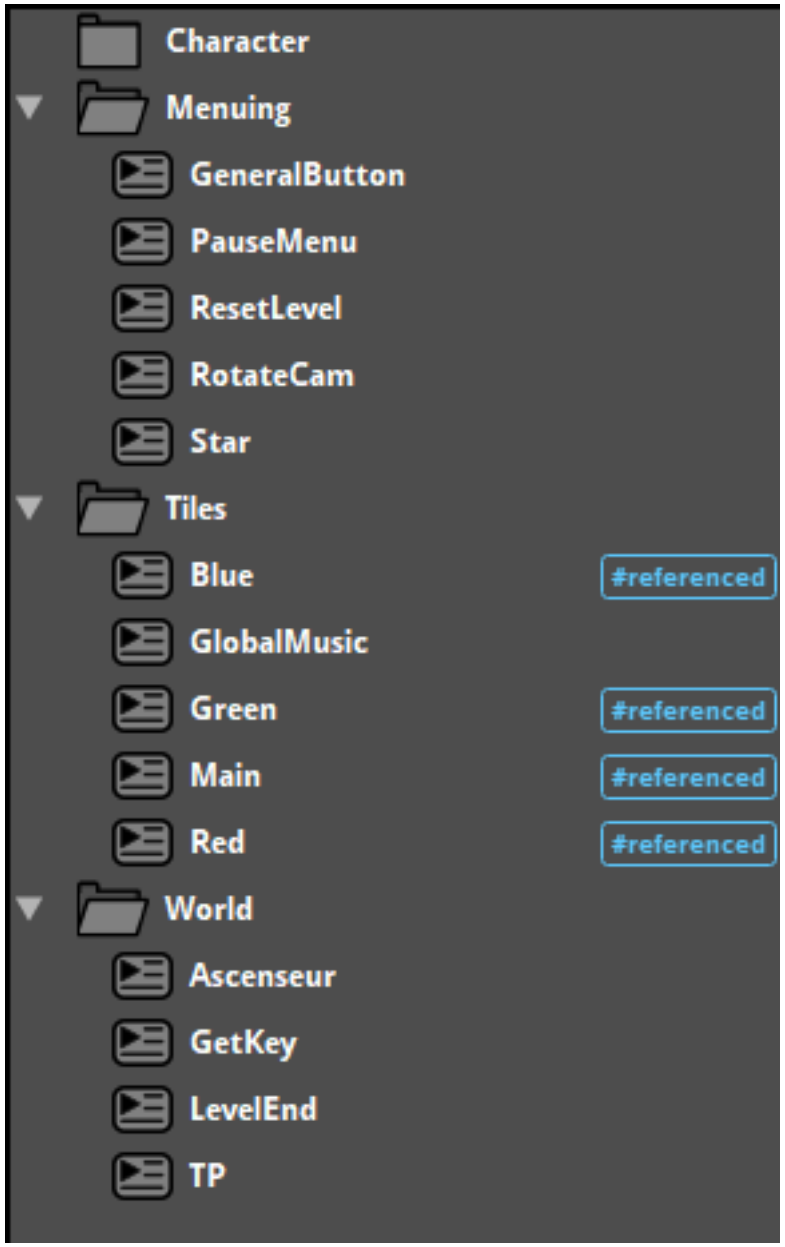


Tableau des sons

		EventName	Priorité	Spécifique / Générique	Boucle/Event	Description	Paramètre / variable	2D / 3D	Condition de Triggers	References	Etat d'avancement					
											État de la tâche	A faire	Créer	A intégrer	Fini	Bugfix
3	world															
4		level start	3	générique	event	pionnn		2d	entrée de niveau		place holder	☑	☐	☐	☐	☐
5		level end	1	générique	event	tu dun		2d	Sortie de niveau		place holder	☐	☐	☐	☑	☐
6		Tempo Tile red	1	générique	boucle	kick		2d	Tempo Tile Rouge up		place holder	☐	☐	☐	☑	☐
7		Tempo Tile blue	1	générique	boucle	basses		2d	Tempo Tile blue up		place holder	☐	☐	☐	☑	☐
8		Tempo TIle green	1	générique	boucle	saxophone		2d	Tempo Tile green up		place holder	☐	☐	☐	☑	☐
9		normal tile	1	générique	boucle	mélodie principale		2d	entrée de niveau		place holder	☐	☐	☐	☑	☐
10		block tp	2	générique	event	zwoop		2d	entrée dans un tp		place holder	☐	☐	☐	☑	☐
11		block ascenseur up	2	générique	event	vuut aiguë		2d	ascenseur up		place holder	☐	☐	☐	☑	☐
12		block ascenseur down	2	générique	event	vuut grave		2d	ascenseur down		place holder	☐	☑	☐	☐	☐
13		Activate leve	2	générique	event	din din ding		2d	activation d'un levier		place holder	☐	☐	☐	☑	☐
14	UI															
15		pause menu	3	générique	event			2d	click menu pause			☐	☐	☐	☑	☐
16		button	3	générique	event			2d	click button			☐	☐	☐	☑	☐
17		reset level	3	générique	event			2d	click reset button			☐	☐	☐	☑	☐
18		camera rotate	3	générique	event			2d	click camera rotate button			☐	☐	☐	☑	☐
19																
20	system															
21		victory screen	3	générique	event	tadaaa		2d	fin de niveau, écran de transition			☐	☐	☐	☑	☐
22		star count	3	générique	event	tin, tin, TIN		2d	décompte étoile			☐	☐	☑	☐	☐
23		all stars	3	générique	event	musique enjouée		2d	le joueur a les 3 étoiles			☑	☐	☐	☐	☐
24		locked content	3	générique	event	tictictic		2d	contenu verrouillé			☑	☐	☐	☐	☐
25		unlocking content	3	générique	event	tic; clack !		2d	débloquer du contenu			☑	☐	☐	☐	☐

Organisation

- Pipeline de production -
- Trello -
- GitHub -

Pipeline de production

Fonctionnement

Pour ce projet de second semestre, le corps enseignant nous impose des jalons avec un certain avancement demandé. L'organisation entre ces étapes nous est laissée libre. La méthode de travail adoptée pour Araka est une méthode agile dont les sprints durent une semaine. Ce rythme est adapté aux jalons courts (1 mois) et au format mobile de notre projet, nous permettant d'avoir des builds jouables toutes les semaines. L'organisation de chaque semaine suit le Pipeline de Production suivant.

Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
Réunion : discussion des tâches à effectuer dans la version de la semaine, rédaction de documents et GD	Production : Programmation et création d'asset graphiques, compositions des sons. Rédaction de documents	Réunion : suivit de l'avancmeent de la version, premiers tests et premiers retours	Production : Programmation et implémentation d'asset graphiques et de sons. Débug et playtest	Production : Débug et playtest	Réunion : avis et retours sur la version suites aux tests, mise à jour de la documentation	Repos

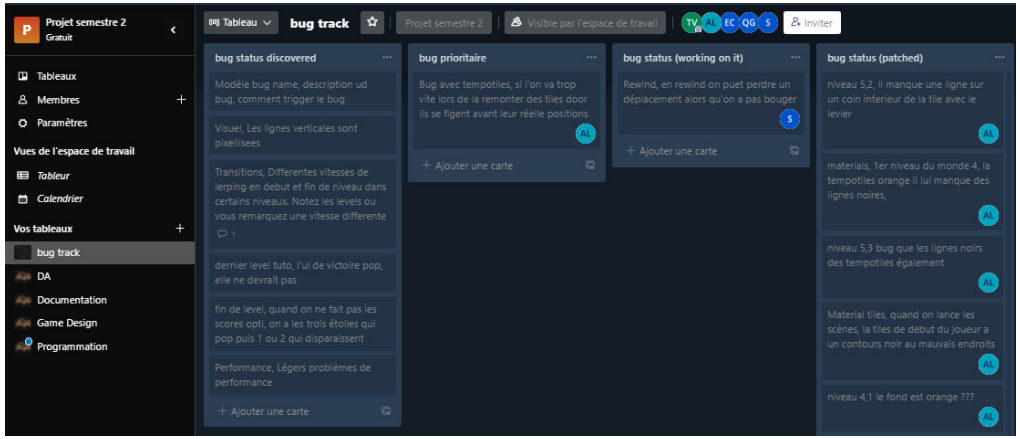
Trello

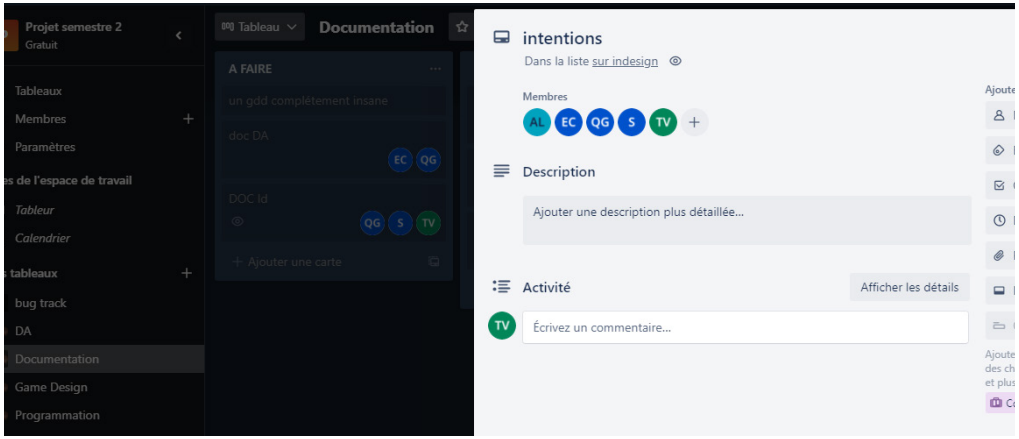
Mise en place

Dès le lancement du projet, un trullo a été mis en place avec un tableau par domaine. Ces tableaux sont structurés par statut de tâche (a faire, en cours, fait) et par qui est sur la tâche. Cela permet de suivre l"avancement du projet point par point et de nous fixer des objectifs pour la semaine.

Fonctionnement

Suite aux réunions du début de semaine, des tâches sont listées dans leurs catégories respectives, vis-à-vis des besoins fixés par les objectifs de la semaine. Les membres du groupe peuvent alors se positionner sur des tâches proposées en fonction de leur rôle au sein du projet. Il est ainsi possible par exemple de venir en soutien d'un autre membre du groupe sur sa tâche s'il est en difficulté.

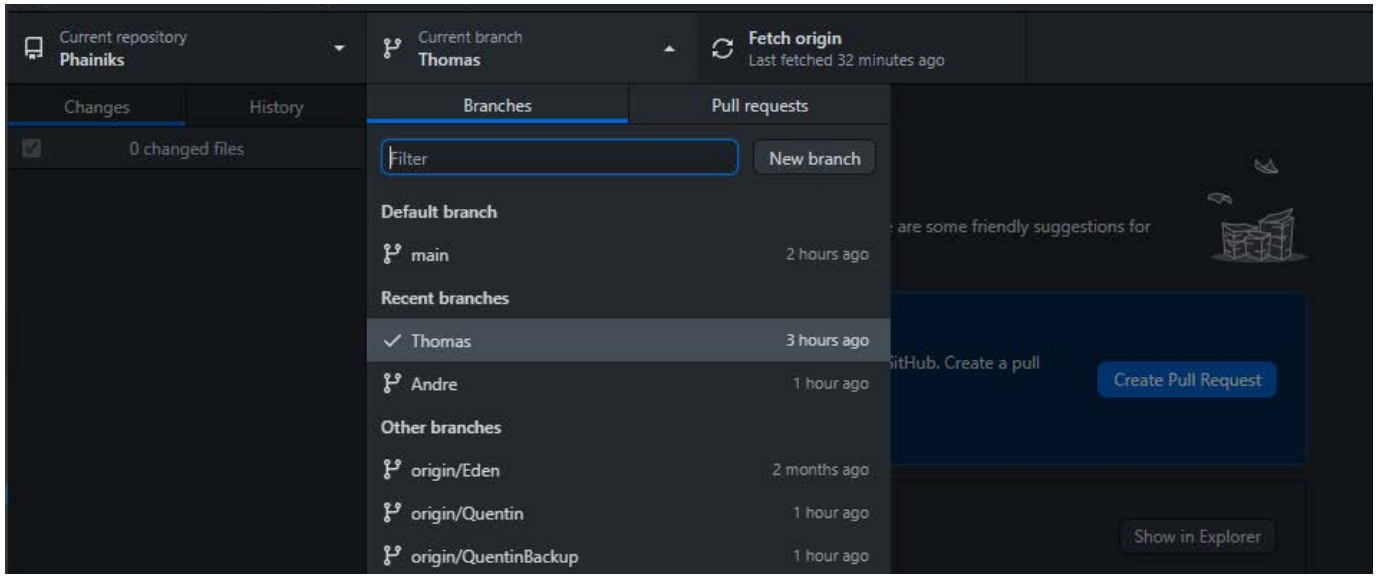




GitHub

Utilisation

Pour pouvoir travailler de manière constante sur le projet et pouvoir mettre en place ce mode de fonctionnement agile, nous avons décidé d'utiliser GitHub pour Araka. GitHub nous permet de travailler sur le même projet Unity et de fusionner nos modifications, mais aussi de pouvoir revenir aux versions précédentes si un problème survient ou si nous ne sommes pas satisfaits des changements du sprint. GitHub n'étant pas un outil parfait, nous utilisons également google drive pour backup nos versions.



ARAKA