

Rapport de Stage de 5ème année
du Cycle Ingénieur à Polytech Paris-Saclay

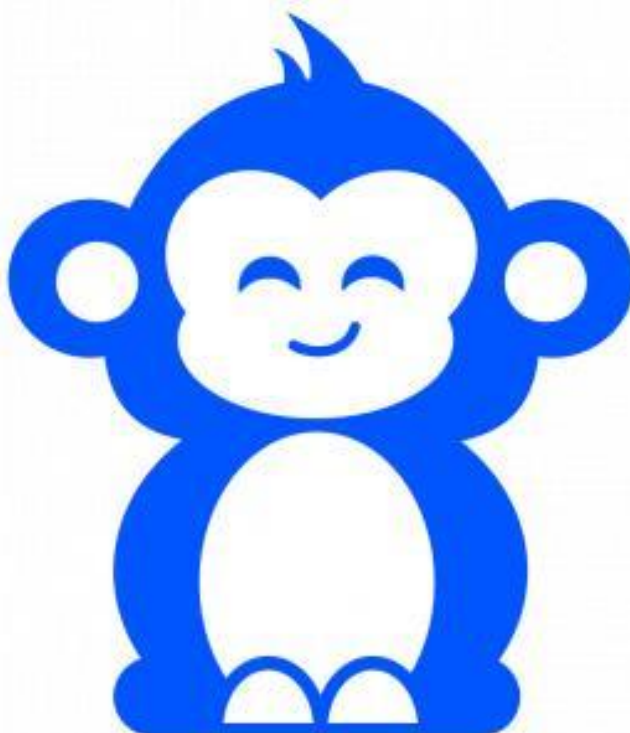
Assistant Ingénieur

Du 7 mars au 26 août 2022

Tuteur école : Joël Falcou

Joel.falcou@universite-paris-saclay.fr

Développeur de Jeux Vidéo 3D Unity



Organisme d'accueil : Blue Monkey Studio

Adresse : 3 rue des Mouettes

Responsable organisme d'accueil : Mathis Delaunay

Téléphone entreprise et Mail : +336 14 74 13 99 mathis@bluemonkey.studio

I. Sommaire

Table des matières

I.	Sommaire	1
II.	Remerciements	2
III.	Résumé.....	3
IV.	Introduction.....	3
V.	Développement :	4
1.	Présentation de l'entreprise.....	4
1.1.	Contexte socio-économique de l'entreprise.	4
1.2.	Ma place dans l'entreprise	4
2.	Présentation de la mission menée	5
2.1.	Contexte de la mission	5
2.2.	Description de la mission.....	6
2.3.	Cahier des charges.....	6
2.4.	Organisation et planification dans le temps.....	8
3.	Compte rendu.....	10
3.1.	La phase d'apprentissage	10
3.2.	L'idéation.....	11
3.3.	Développement d'une solution de gestion des applications automatique	14
3.4.	Développement du premier prototype (Frog Run)	15
3.5.	Développement du deuxième prototype (Flip and Snap)	19
3.6.	Développement du troisième prototype (Shape Together).....	21
3.7.	La première itération sur un prototype (Flip and Snap).....	23
3.8.	Retour sur l'automatisation de gestion d'application.....	25
3.9.	Développement du dernier prototype (Light the Fuse)	26
VI.	Bilan de la mission	30
1.	Résultats	30
1.1.	Analyse des résultats.....	30
1.2.	Gestion des contraintes.....	35
1.3.	Méthodologie et Choix techniques	35
2.	Retour d'expérience	36
2.1.	Recul sur les Résultats mission	36
2.2.	Difficultés.....	36
2.3.	Apports personnels	37
2.4.	Aspect sociaux-environnementaux de ce stage	39

2.5.	Futur immédiat	40
2.6.	Perspective sur deux ou trois ans	40
2.7.	Mon cycle ingénieur	40
VII.	Conclusion	41
VIII.	Lexique et sources	41
1.	Vocabulaire de l'hypercasual	41
2.	Logiciels et plateformes utilisées au cours du stage	45
IX.	Bibliographie.....	47
X.	Annexe.....	48

II. Remerciements

Avant de débiter ce rapport de stage, je souhaiterai remercier en premier lieu Harvey et Mathis, les fondateurs de Blue Monkey Studio, pour m'avoir accueilli si chaleureusement au sein de leur entreprise durant ces six derniers mois. Je souhaite remercier également mes amis et camarades de projets qui m'ont poussé il y a de cela cinq ans à travailler avec eux sur des prototypes de jeux vidéo, et ont su me faire découvrir et apprécier ce monde au point que je finisse par vouloir en faire mon métier. Je remercie enfin toute l'équipe de Polytech pour la qualité des cours dispensés, et l'accompagnement que j'ai pu recevoir au cours de ces années d'études.

III. Résumé

Du mois de mars à la fin du mois d'août, j'ai réalisé un stage ingénieur au sein de l'entreprise Blue Monkey Studio. J'ai pu au total participer au développement de quatre jeux sur le moteur de jeu Unity, et développer divers scripts, qui n'ont pas d'utilité pour le jeu vidéo en lui-même mais pour l'application dans sa globalité. Les quatre jeux développés font tous partie du domaine de l'hypercasual, mais sont néanmoins tous très distincts, par leurs mécaniques et leur aspect visuel. J'ai eu la chance de participer activement à toutes les phases de la création d'un jeu vidéo sur la quasi-intégralité des mes projets, de l'idéation du concept du jeu à sa publication sur les **stores**, en passant évidemment par son développement en C#. J'ai également pu en apprendre plus sur la relation entre un studio de développement et son éditeur attiré. Ce stage m'a également permis, contrairement à mes précédents stages, à non plus réaliser des tâches au sein d'un projet, mais à participer à l'intégralité des tâches d'un projet, et ainsi l'accompagner jusqu'à son achèvement.

IV. Introduction

La fin de mon cursus ingénieur approche à grands pas. Il ne me reste plus qu'une étape à surmonter avant la fin de l'année : le stage de fin d'études. C'est dans l'entreprise Blue Monkey Studio que j'ai eu l'opportunité d'effectuer un stage de six mois dans un domaine que j'adule tant : le jeu vidéo. Au cours de ce stage j'ai pu développer des jeux pour mobile aussi bien que mes compétences professionnelles, auprès de spécialistes du domaine. J'ai pu découvrir un pan entier de mon domaine de prédilection que je ne connaissais que trop peu, et vois désormais le développement de jeux vidéo avec un œil nouveau.

Cette nouvelle expertise et ce regard nouveau, ce sont exactement pour ces raisons que j'ai recherché un stage dans le domaine du jeu vidéo à tout prix. Je cherche à l'avenir à obtenir une place dans une entreprise de ce domaine, tout en conservant un niveau ingénieur. Blue Monkey Studio, étant spécialisé dans la conception de jeux mobiles, constitue un très bon point d'entrée dans ce secteur, surtout pour une personne comme moi n'ayant pas suivi un cursus spécialisé dans l'art vidéo ludique.

Je vais dans ce rapport décrire l'ensemble du travail que j'ai pu réaliser en détail, les résultats de ce travail, et les diverses expériences que je me suis forgées lors de ces six derniers mois.

Petit avant-propos : tous les mots possédant une couleur particulière sont définis dans la section Lexique. Le bleu signifie un mot du vocabulaire des jeux **hypercasual**, quand le vert signifie un outil comme **Unity**.

V. Développement :

1. Présentation de l'entreprise

1.1. Contexte socio-économique de l'entreprise.

Blue Monkey Studio est une société par actions simplifiée française créée il y a un an et demi. Elle est spécialisée dans le développement de jeux vidéo sur mobile. Elle a pour éditeur attiré l'entreprise française Voodoo. Ses jeux ont déjà été téléchargés plus de vingt millions de fois dans le monde entier, avec actuellement une communauté active s'élevant à plus de trente milles joueurs. Son siège social se situe au 3 rue des mouettes à Mont-Saint-Aignan, en Seine-Maritime.

L'entreprise est parvenue au bout de neuf mois après sa création à développer un jeu qui s'est téléchargé plus de dix millions de fois, Flex Run 3D.

L'entreprise a été fondée par Harvey Roberts, son président, et Mathis Delaunay, son directeur général. Elle a depuis compté parmi ses membres un graphiste et de nombreux stagiaires et alternants.

L'activité de cette entreprise consiste en la conception de jeux vidéo hypercasual dans un premier temps en phase de prototype, puis le prototype est testé sur le marché, et s'il son potentiel s'avère, le prototype est approfondi en une version beta, qui si elle rencontre un franc succès, deviendra un jeu à part entière. L'objectif est de trouver le plus d'idées de jeu possible, d'en développer des prototypes le plus vite possible et de ne conserver que les meilleurs.

Les jeux produits ont pour objectif de divertir les clients, que nous appelons tout sobrement les joueurs. Ils doivent passer un moment agréable en jouant à nos jeux, afin de décompresser d'une dure journée au travail, pour faire passer le temps dans les transports en commun ou tout simplement pour se détendre et lâcher prise, pour s'évader.

Le marché sur lequel Blue Monkey Studio se positionne est celui du jeu vidéo hypercasual. C'est un marché sur lequel la concurrence est extrêmement rude. De très nombreux studios développent de tels jeux, et cherchent tous à créer un jeu qui sera mis en avant sur les stores pour attirer plus de joueurs et ainsi faire de meilleures recettes. On peut citer parmi les studios concurrents h8games ou Cassette. Mais tous les studios ne sont pas nécessairement en grande concurrence sur ce marché. Les entreprises se contentent la majeure partie du temps de développer les jeux, et sont sous l'égide d'un éditeur, qui va faire des études de marché approfondies, dispenser des conseils et suggestions aux studios de développement et s'occuper de la visibilité des jeux édités. Ainsi, les jeux de divers studios se feront moins d'ombre les uns les autres s'ils sont édités par la même entreprise. Les principaux éditeurs dans le domaine de l'hypercasual sont Voodoo, éditeur de Blue Monkey Studio et premier du marché depuis quatre ans, Lion Studio, SayGames ou encore Crazy Labs, mais il en existe bien d'autres.

1.2. Ma place dans l'entreprise

J'ai rejoint l'équipe de développement de cette entreprise en tant que stagiaire, afin de la soutenir dans son processus de conception de jeu sur toutes les phases du développement. Blue Monkey Studio pourra ainsi développer plus de prototypes, les développer plus rapidement, et donc augmenter ses chances de créer un jeu à très fort succès.

En termes de hiérarchie, il a assez peu de choses à dire. Étant donné la taille de l'entreprise, mes seuls supérieurs hiérarchiques sont les deux fondateurs de l'entreprise, parmi lesquels mon tuteur de stage en entreprise (Mathis Delaunay). Les autres employés sont au même étage que moi sur la pyramide hiérarchique.

2. Présentation de la mission menée

2.1. Contexte de la mission

L'objectif de la mission se résume en une courte phrase : concevoir des jeux vidéo mobiles pour satisfaire le besoin de divertissement de monsieur et madame tout le monde, et pas nécessairement de joueurs aguerris.

Le développement se fait en premier lieu sous la forme d'un prototype réalisé en un mois environ par une équipe de deux à quatre personnes, en fonction de la difficulté estimée du projet et de son état d'avancement. Une fois le jeu terminé, le studio ajoute au projet des scripts fournis par l'éditeur, et publie son jeu sur l'App Store, et crée une page pour le jeu sur **Facebook Developer** pour en faire la publicité. L'éditeur récupère ensuite ces données et prend le relais. Suite à l'analyse de ces données, il choisit si le jeu vaut le coup d'être approfondi jusqu'à une version définitive. L'objectif à long terme du studio est donc de produire des jeux qui feront un carton, mais à plus court terme, comme durant les six mois qu'a duré mon stage, l'objectif est de réaliser un prototype avec un potentiel suffisant pour que l'éditeur décide qu'il est rentable de poursuivre son développement.

L'intégralité des membres de Blue Monkey Studio participe au moins de façon mineure à l'intégralité des prototypes. Une personne est à l'origine d'une idée, mais tout le monde en débat pour qu'on présente à l'éditeur la meilleure version possible de l'idée. Deux à quatre personnes interviennent directement dans le développement du jeu, au moins un graphiste et le reste du personnel sont des développeurs. Tout au long du développement, des vidéos de **gameplay** et des exécutables sont mis à disposition des collègues ne participant pas directement à la réalisation du jeu pour donner un premier avis extérieur. Ensuite, l'un des développeurs se charge de la publication du prototype, puis tout le monde est affecté à la conception d'un nouveau prototype et la boucle est bouclée.

Au total, au fil du stage, j'ai pu collaborer avec un graphiste et cinq développeurs.

2.2. Description de la mission

Mon rôle au cours de ce stage est multiple. Dans l'ensemble, je suis intervenu sur toutes les étapes de la conception de jeux vidéo hypercasual.

La première étape est l'idéation. Le but ici est de trouver des idées nouvelles de jeux à développer selon les contraintes imposées par le domaine de l'hypercasual ainsi que les préférences de l'éditeur.

La seconde mission est le développement de ces idées, en prototypes de jeux dans un premier temps. Le développement peut être divisé en plusieurs étapes qui seront détaillées plus tard dans ce rapport : établissement du cahier des charges, développement des mécaniques de jeu principales, conception des niveaux (le **level design**), phase de tests.

La troisième étape et dernière du cycle de développement d'un jeu hypercasual est l'intégration des sdk de l'éditeur et la distribution du jeu sur les stores, ainsi que la diffusion de sa publicité. Une mission supplémentaire m'a été assignée, la recherche et le développement de solutions pour automatiser le plus possible les tâches de cette dernière phase de développement de jeu.

2.3. Cahier des charges

Le cahier des charges associé à cette mission est plutôt simple. Toutes les phases de développement se succèdent chronologiquement sans aucun retour en arrière, mais un projet peut potentiellement être mis en pause et report. Tout le long du projet, nous utilisons des principes, des méthodes de travail et des technologies très spécifiques, communes à toute l'équipe.

À la suite d'un brainstorming, nous devons noter notre idée, qu'elle soit aboutie ou non, sur une fiche postée sur **Notion**. Cette fiche rassemble tous les éléments essentiels à la compréhension globale de l'idée. Elle recense une description générale du jeu imaginé, les contrôles (par exemple s'il faut juste taper du doigt ou le faire glisser sur l'écran), le type de caméra (généralement, vue du dessus ou vue de type **runner**), les conditions de victoire et de défaite, certains détails qui différencient l'idée de jeux dans le même genre déjà existants, des références de gameplay et artistiques, un **thème**, les sensations que l'on cherche à provoquer chez le joueur et des schémas explicatifs. Une fiche terminée définit avec précision le jeu que l'on cherchera à développer si l'éditeur donne son accord.

Toutes les personnes travaillant chez Blue Monkey Studio ont accès en permanence à l'intégralité de ces fiches, qui sont classées différemment selon l'aboutissement de l'idée ou l'avancée de son développement, si elle a été retenue etc... Lorsqu'une fiche vient d'être postée, n'importe qui dans l'équipe peut donner son avis, des éléments d'amélioration ou encore sa vision de l'idée en commentaire.

Une fois une idée sélectionnée, nous devons avant toute chose établir un diagramme des classes. Outre le fait de donner une vision très générale du projet, d'exprimer sa faisabilité et même une estimation de sa complexité, ce diagramme nous permet avant tout d'estimer quelles sont les classes et interfaces que l'on pourra réutiliser de projets précédents, mais aussi les nouvelles classes qui pourront éventuellement être réutilisées à l'avenir sur de nouveaux projets. Cette notion de réutilisation du code est très importante en développement de jeux hypercasual, puisque pour une trentaine de jeux développés, un seul sera une réelle réussite, ce que l'on appelle un **hit**. Il faut donc

pouvoir développer le plus de prototypes possible en le moins de temps possible. Et la réutilisation de code représente un gain de temps considérable pour ce genre de projet, en particulier quand il s'agit d'algorithmes et de structures complexes comme le [pathfinding](#) et les systèmes utilisant une grille.

Enfin, le développement en lui-même est très encadré, et doit suivre des règles précises. Les premières règles sont des principes de développement : le suivi des [principes SOLID](#), ainsi que le développement en [MonoBehaviour](#). Ces principes sont essentiels à la rédaction de code réutilisable, en plus de faciliter sa clarté, sa compréhension, son adaptabilité à d'autres classes ou interfaces et son intégration dans une structure de code.

La seconde règle est la manière d'appliquer ces principes. Chaque nouveau projet est accompagné d'une [scène](#) par défaut contenant toutes les composantes nécessaires au développement d'un jeu, une structure pour ranger les différentes composantes du jeu, ainsi que des [Bundles](#) contenant les classes réutilisables développées lors de précédents projets. Le jeu doit être décomposé en plusieurs mécaniques. Une mécanique est un ensemble de trois types de classes qui vont ensemble permettre au jeu de fonctionner. Chaque classe créée lors du projet devra donc être (à quelques rares exceptions près) de l'un des trois types suivants :

-Une action : une action est un résultat particulier et distinguable qui peut se produire au cours du jeu. Par exemple, un déplacement d'une position à une autre, une rotation, un changement de couleur ou encore la destruction d'un objet sont des actions. Une action, contrairement à ce que son nom peut indiquer, a tendance à être passive vis-à-vis du jeu. De manière générale, elle est la conséquence des mécaniques, l'ensemble des réactions indirectes.

-Un événement : un événement correspond à un déclencheur. Il va provoquer une action particulière dans le jeu. Par exemple, la collision entre deux objets A et B peut être considérée comme un événement, ou encore l'[input](#) d'un utilisateur. L'événement constitue la cause d'une mécanique. Ce sont les réactions directes de l'utilisateur avec le jeu ou les réactions indirectes des actions. A noter qu'une action peut également constituer un événement, par exemple une animation est une action, mais la fin de cette animation peut-être un événement qui déclenchera une autre action.

-Un bridge : Le bridge est le cœur de la mécanique, il fait le lien direct entre l'action et l'événement. Il existe donc autant de bridge que de combinaison action/événement possible. Par exemple, tourner quelque chose en le touchant est un bridge au même titre que faire apparaître un boulet quand un canon est déclenché.

Dans de rares cas, nos classes peuvent ne pas être de ces trois types. Par exemple lorsqu'on a besoin de [managers](#), bien qu'il soit très peu recommandé d'en créer de nouveaux, puisque tous ceux préexistants dans la structure d'un nouveau projet sont censés nous permettre de pouvoir développer n'importe quel jeu dans son intégralité. En revanche, un exemple plus approprié pourrait être les classes de type calculateur. Ce sont des classes dont le seul rôle est d'effectuer des calculs, comme leur nom l'indique. Une classe faisant des calculs matriciels doit être développée sous la forme d'un calculateur. Il en va de même pour une classe qui va analyser les caractéristiques des cases dans une grille, comme l'état vide ou plein d'une case particulière ou d'un ensemble de cases. Le calculateur va fournir des informations qui vont potentiellement déclencher des actions. Dans notre dernier exemple, une case vide pourrait provoquer une action permettant de faire apparaître un objet pour la remplir. Le calculateur pourrait pourquoi pas chercher un chemin parmi les cases pour permettre à l'objet généré de rejoindre sa case vide via l'application d'un déplacement. Un autre exemple que nous utilisons dans chaque jeu est l'ensemble des classes de GameLogic. Il est possible que dans notre jeu,

un objet puisse avoir de nombreuses interactions possibles, ce qui implique une très grande quantité de bridges. Or, les bridges, contrairement aux événements et aux actions, sont très peu réutilisables. Il peut donc être très rébarbatif de créer le nombre de bridges nécessaire pour faire fonctionner notre mécanique et de les ajouter un à un au **gameObject** associé. La GameLogic permet de condenser l'ensemble de ces bridges en un seul fichier.

Enfin, les dernières règles à appliquer sont les technologies à utiliser. Tout le monde travaille avec les mêmes outils et les mêmes versions de ces outils quand cela est possible par souci de conformité. Ainsi, les mises en page des codes ne sont pas cassées systématiquement à chaque fois que quelqu'un récupère des fichiers de code sur **Github**, et personne n'a de problème de compatibilité pour exécuter son code.

Nous éditons tous notre code via **JetBrains Rider** 2022.1.2, et utilisons le moteur de jeu Unity, d'abord dans sa version 2020.3.30f1, puis dans sa version 2021.3.5f1 depuis que l'éditeur a mis à jour sa sdk pour la rendre compatible avec des versions stables plus récentes d'Unity. Nous utilisons tous **Sourcetree** pour gérer nos projets sur Github, et faisons des push réguliers chaque fois qu'on dispose d'une version stable de notre projet avec une nouvelle fonctionnalité ou une refonte graphique. La structure de branches sur Github est d'ailleurs elle-même imposée.

2.4. Organisation et planification dans le temps

L'élément central d'une bonne planification est bien entendu la réunion des membres des équipes concernés par le projet. Nous avons très régulièrement des réunions sur **Google Meet** au cours d'un projet. Ces réunions ont divers objectifs, diverses durées et diverses planifications :

Les réunions les plus importantes sont les réunions dites « du lundi », en raison de la journée durant laquelle ces réunions ont lieu chaque semaine. Les réunions du lundi permettent de faire le point sur tous les projets en cours, l'avancement de chacun, et de planifier les objectifs à atteindre à la fin de la semaine. On profite aussi de ces réunions pour clarifier certains points si besoin sur les projets en cours, et permettent d'informer tout le monde des nouveautés chez Blue Monkey Studio, qu'il s'agisse des absences pour congé, de la disponibilité de nouveaux Framework ou encore d'organisation d'événements (par l'entreprise ou par l'éditeur). Tous les membres de Blue Monkey Studio assistent à ces réunions qui durent en moyenne une heure et demie.

Il existe une autre réunion hebdomadaire à laquelle tout le monde est convié qui a lieu les jeudis. Il s'agit des « calls des idées ». Chaque semaine, chaque membre de l'équipe consacre deux heures à la recherche d'idées de nouveaux jeux, aux horaires qui lui conviennent le mieux avec les techniques de brainstorming qui lui conviennent le mieux. Le call des idées a pour but de discuter des idées que chacun a noté sur une fiche publiée sur Notion. Certaines sont retenues, d'autres adaptées, d'autres encore mises de côté en attendant une amélioration drastique (comme la découverte d'un thème spécifique pouvant améliorer la **rétenion** par exemple). Les dernières sont tout simplement rejetées car nous estimons qu'elles n'ont pas assez de potentiel, ou qu'elles ne correspondent pas aux standards des jeux hypercasual ou de l'éditeur. Les idées sélectionnées sont ensuite présentées à un représentant de l'éditeur, qui va donner son avis sur l'idée. Cet avis n'est jamais un refus catégorique, puisqu'un tri a préalablement été effectué par l'entreprise à l'origine de l'idée. Cet avis peut être un feu vert si l'idée est novatrice, innovante et plaît au représentant, une suggestion d'adaptation si le

représentant estime qu'un point de l'idée ne répond pas suffisamment aux critères de l'hypercasual, une mise en garde si le concept est recevable mais ne plait pas plus que ça au représentant ou une suggestion d'abandon de l'idée si celle-ci ressemble trop à un projet passé qui n'a pas fonctionné. Ces réunions pour débattre sur les idées et sélectionner les meilleures peut durer jusqu'à 2h, bien que sa durée exacte soit très variable.

Une autre réunion importante, mais mensuelle cette fois-ci, vise à faire un point sur le déroulement du stage et l'organisation de l'entreprise (du moins dans mon cas c'est un point sur le stage). Chaque membre de Blue Monkey Studio rencontre les deux fondateurs de l'entreprise afin de discuter du déroulement des opérations, des éventuels axes d'amélioration dans les méthodes de travail, de la manière dont les tâches sont réparties, et des potentiels soucis que chacun peut rencontrer.

Au démarrage du développement d'un nouveau jeu, toute l'équipe qui va intervenir sur le jeu se rassemble pour définir la répartition des tâches.

Enfin, les dernières réunions ne sont pas organisées à l'avance. Il s'agit de réunion entre deux ou trois personnes au maximum, qui ont pour objectif d'éclaircir certains points sur la structure du projet, sur l'organisation des assets, sur la fusion de deux versions du projet qui ne se serait pas déroulée comme prévu... On peut qualifier ces réunions d'« impromptus essentiels » lorsqu'on télétravaille.

Pour chaque idée de jeu choisie pour être développée en version prototype, un **Kanban** est associé à sa fiche sur Notion. Ce Kanban est accessible à tous les employés de Blue Monkey Studio, et recense la liste des tâches à accomplir pour réaliser le jeu. Parfois même, si besoin, les tâches sont découpées en sous tâches, pour diviser au mieux le travail. Chaque tâche peut être attribuée à n'importe quel membre de l'équipe de développement sur le Kanban. Ainsi, chacun sait qui travaille sur quelle fonctionnalité, et sais donc à qui s'adresser en cas de besoin et n'empiète pas sur le travail des autres. Chaque tâche peut être rangée dans une sous-catégorie du Kanban. Généralement, tous les Kanban contiennent des catégories comme « To do », « doing » ou encore « done », mais on peut rajouter des catégories supplémentaires au besoin, par exemple pour une tâche initialement prévue mais qui a été annulée pour une certaine raison.

Ni l'éditeur, ni l'entreprise n'impose de délai strict pour la création de prototypes. Le but est simplement d'en créer le plus possible le plus vite possible. En général, un prototype complet prendra entre trois et cinq semaines à être développé. Parfois il peut durer plus longtemps, notamment si on souhaite créer une nouvelle itération de ce projet, c'est-à-dire d'en faire une nouvelle version qui diffère légèrement de la précédente pour tester l'impact de ce changement sur le marché. Si un projet commence à mettre trop de temps à être développé, une réunion est organisée pour tenter de cerner les causes de ce retard, et le projet est ensuite généralement mis en pause, le temps de trouver une solution pour accélérer le développement.

Les seules fois où les diverses phases de conception des jeux sont soumises à une contrainte temporelle stricte sont lorsque l'éditeur organise des concours. Dans ces cas-là, il est évident que si nous participons à ces concours, l'éditeur fixe une date limite aux entreprises qu'il édite pour achever les projets soumis au concours. Cette situation est intervenue deux fois uniquement lors de mon stage.

3. Compte rendu

3.1. La phase d'apprentissage

Lors de mon arrivée en entreprise, la toute première mission qui m'a été affectée a été la découverte du monde de l'hypercasual. Le but était de passer une semaine à étudier des vidéos sur le sujet, afin d'être opérationnel le plus vite possible. Ces vidéos ont été mises à la disposition des entreprises de développement de jeux vidéo par l'éditeur sur son site, la **Voodoo Academy**. Le domaine de l'hypercasual est un sous domaine du jeu vidéo qui est soumis à des règles qui lui sont propres, bien loin des connaissances que je possédais déjà sur la création de jeux vidéo plus classiques. Les règles générales les plus importantes, au point d'être respectées impérativement par tous les jeux du genre, sont :

- Le public visé n'a aucune limitation. Tous les jeux s'adressent aussi bien aux joueurs expérimentés et aux habitués des réseaux sociaux, qu'aux parfait débutants qui n'ont même jamais touché à un téléphone. De manière générale, le jeu doit être compréhensible et avoir un thème parlant même pour les personnes âgées, qui ne sont généralement pas les personnes visées sur les marchés des hautes technologies. Exceptionnellement, seuls les enfants peuvent constituer une limite au public visé, étant donné que tous les jeux vont recevoir une limitation d'âge minimum (au minimum déconseillé aux enfants de moins de trois ans). Cette limitation aux enfants peut monter jusqu'aux dix-huit ans nécessaires pour jouer en raison de la présence d'armes à feu ou de violence explicite par exemple, bien que de tels jeux soient très rares dans ce domaine. Cette règle découle du fait que les revenus engrangés par les jeux hypercasual se fait majoritairement selon la publicité présente au lancement du jeu. Plus nombreux seront les utilisateurs, plus les publicités seront visionnées. Et plus les publicités ont été visionnées, plus le jeu rapportera de l'argent. On dit que le jeu doit être mass market. Pour rendre le jeu le plus mass market possible, il faut éviter les thèmes de niche (comme le football ou la guerre par exemple) et que le déroulement du jeu soit intuitif à l'extrême que ce soit au niveau des contrôles (S'il y a un bouton, c'est qu'on peut appuyer dessus et s'attendre à une réaction) ou aux diverses réactions de l'environnement (Si deux objets se percutent, ils ne se traversent pas). Au contraire, les jeux vidéo plus classiques vont s'adresser aux joueurs confirmés, maîtrisant les codes du domaine. Et bien souvent, le public visé sera même une niche parmi ce public plus large, comme les jeux FIFA s'adressent aux joueurs également fans de football ou ayant envie de jouer avec des amis dans une même pièce comme en ligne.

- Les contrôles du jeu doivent être extrêmement simples et limités, en général pas plus d'un input par jeu. De plus les contrôles doivent être exercés sur une surface très spécifique du téléphone, à savoir tout l'écran excepté la partie supérieure gauche. Il faut également éviter les contrôles sur la partie supérieure de l'écran. Ces restrictions proviennent d'un phénomène propre aux jeux hypercasual : ce sont de petits jeux auxquels les joueurs vont le plus souvent jouer après le travail, dans les transports en commun notamment. Et ceux-ci risquent alors de n'avoir qu'une seule main disponible pour jouer. Les jeux plus classiques eux, sont développés sur des plateformes munies de manettes ou de claviers permettant l'ajout de bien plus de contrôles de la part du joueur.

- Les jeux doivent être composés de niveaux très courts (une minute trente en moyenne), proposant de nouvelles fonctionnalités ou mécaniques régulièrement. L'objectif ici est de maintenir l'intérêt du joueur, car une fois de plus, un joueur qui reste longtemps est un joueur qui aura vu beaucoup de publicités. Toutefois, la notion de niveau peut varier selon le jeu. Il peut s'agir d'un puzzle spécifique à

résoudre, d'une session de jeu dans un environnement aléatoire, d'une course ... Cette division en niveaux est très variable dans les jeux plus classiques, et parfois même inexistante.

- Les jeux doivent être à leur façon très satisfaisants. Ce sentiment de satisfaction recherché permet de créer l'addiction chez le joueur, qui reviendra jouer au jeu après une partie pour retrouver ce plaisir qu'il a ressenti lors de sa précédente session. Bien souvent on associe les jeux hypercasual à l'**ASMR**, comme c'est le cas pour les jeux de type Do It Yourself qui permettent de créer librement une poterie, un dessin ou n'importe quelle œuvre imaginable. Mais la satisfaction peut être implémentée de plein d'autres façons : des sons (comme une musique apaisante), des sensations haptiques (les vibrations du téléphone à l'intervention d'un événement particulier), des visuels alléchants (des fruits juteux, un feu qui se propage, des explosions à tout va), ou une mécanique faite pour se passer les nerfs dessus (détruire des bâtiments, faire tomber des bonshommes en **ragdoll**), ou pour donner l'impression d'être meilleur que les autres en dérogeant aux règles (couper à travers champs lors d'une course). Même sans forcément être un jeu ASMR, les jeux hypercasual vont souvent s'en inspirer pour retrouver dans leur gameplay des sensations satisfaisantes, comme c'est notamment le cas du slime. Enfin, cette satisfaction se retrouve aussi dans les récompenses que le joueur recevra s'il joue bien (des combos affichés de partout, un score qui pulvérise les attentes, des cosmétiques pour son personnage ou encore des niveaux bonus). Un jeu vidéo classique peut très bien prendre une direction totalement et axer tout son gameplay sur la frustration du joueur.

En plus de ces règles générales à respecter à tout prix si l'on souhaite réaliser un hit, l'éditeur proposait dans ses vidéos des règles supplémentaires en ce qui concerne la création de jeu. Par exemple, l'éditeur ne conseillera quasiment jamais la conception de jeux de voiture car c'est un domaine qui a certes fonctionné parfois, mais qui est trop de niche et donc trop risqué. De même, elle déconseille très fortement le développement de jeux en deux dimensions. Même si les mouvements des divers objets du jeu se font sur un plan, lesdits objets et les décors doivent être en trois dimensions. Ceci viendrait du fait que les personnes qui ne sont pas familières avec les jeux vidéo se repèrent mieux en évoluant dans un univers en trois dimensions qu'en deux dimensions.

Au cours de ces vidéos explicatives, l'éditeur offre aussi des nombreux conseils pratiques, que ce soit en phase d'idéation ou en phase de développement. Les meilleurs exemples de ces conseils sont les méthodes pour forcer l'idéation et les techniques pour réaliser un code le plus réutilisable possible, étant donné l'importance bien plus nette de la réutilisation de code dans l'hypercasual comparé aux autres domaines du jeu vidéo.

3.2. L'idéation

À la suite de cet apprentissage, j'étais enfin prêt à imaginer des concepts de jeu adaptés à l'hypercasual. Le hasard des choses a fait que Voodoo a organisé à ce moment-là un concours d'idéation. Le but du concours est assez simple, chaque entreprise propose au maximum trois idées de jeu hypercasual, et ces idées vont être triées selon plusieurs critères, dont la correspondance aux critères de l'hypercasual, l'innovation de la mécanique et le potentiel sur le marché actuel. Les dix meilleures idées sélectionnées permettront aux entreprises qui les ont soumises de remporter un prix, d'un montant qui varie selon la place exacte obtenue par le projet. Après ce premier concours d'idéation, un autre concours sera organisé, lui aussi avec des prix en récompense pour les meilleurs, mais faisant intervenir cette fois les prototypes des jeux présentés lors du premier concours.

Blue Monkey Studio a décidé de participer à ce concours, et nous a proposé, à d'autres stagiaires et moi, de rechercher le plus d'idées possible afin de sélectionner les meilleures et de les présenter au concours.

Ainsi ont débutées mes plus grosses séances de brainstorming. Il y en a eu bien d'autres par la suite, mais elles ne prenaient qu'une heure à chaque fois et se faisaient en parallèle d'autres projets. J'ai passé une semaine complète dédiée à la recherche d'idées, entrecoupées de réunions diverses.

J'ai pour mes séances de brainstorming appliqué plusieurs méthodes afin d'arriver à la conception de bons concepts avec un haut potentiel.

La première méthode a été l'étude du marché de l'hypercasual. Blue Monkey Studio a développé une application pour cela qui s'est avérée fort pratique. Cette application, appelée Brainhit, permet de visualiser l'ensemble des prototypes et jeux récemment réalisés, et de les réunir par catégorie pour savoir si une tendance se développait. Les effets de modes sont en effet très importants à discerner, car ils peuvent être la source d'un hit si le gameplay est encore à la mode, ou un échec s'il est développé trop tard. Par exemple, les jeux de stacking, des jeux dont l'élément central est l'empilement de pièces à récupérer en très grand nombre et leur dépilement servant la mécanique principale du jeu, ont été très à la mode il y a deux ans, mais fonctionnent moins bien aujourd'hui. Parfois, en ne voyant que des images des prototypes développés, je peux imaginer une mécanique qui correspondrait, et si cette mécanique n'est pas réellement celle que le prototype propose, je note l'idée sur un document à part, je vérifie que cette idée n'a pas déjà été exploitée, et je l'adapte pour être le plus satisfaisant possible.

Une deuxième méthode, que l'application Brainhit permet également de réaliser, est de fusionner plusieurs concepts. L'idée est de prendre deux concepts qui fonctionnent, qu'il s'agisse d'un thème ou d'une mécanique, et de chercher un jeu qui pourrait implémenter ces deux concepts. Assez souvent, cette méthode ne permet pas de trouver un concept satisfaisant, mais lorsqu'on en trouve un, il s'agit souvent d'un concept très solide car innovant, utilisant des mécaniques familières aux joueurs mais exploitées de façon différente.

Une troisième méthode est l'exploitation de connaissances que j'ai dans le domaine du jeu vidéo en général. Ayant beaucoup joué à divers jeux et en étant déjà passé par de multiples phases d'idéation pour créer des jeux en tous genres, je connais beaucoup de mécaniques diverses adaptables à l'hypercasual, et parfois même des mécaniques jamais vues sur le marché avec un haut potentiel, comme ça a été le cas avec ma première idée basée sur le déplacement grâce à un grapin élastique et collant. J'ai pu trouver bien des idées de jeu via cette méthode lors de cette semaine de brainstorming, mais malheureusement, à long terme elle s'est avérée très limitée, car la majeure partie des mécaniques existantes dans les jeux traditionnels ne sont pas adaptables à l'hypercasual pour diverses raisons, ou alors ne sont pas assez satisfaisantes pour être retenues.

J'ai été amené à tester beaucoup de jeux hypercasual lors de cette semaine, en ayant un regard très critique sur les jeux. L'idée était de me permettre de comprendre mieux encore la substance du jeu hypercasual en ce début de stage, tout en tentant de trouver des axes d'amélioration pour ces jeux que nous pourrions exploiter dans notre propre version du jeu. Par exemple, un jeu avec un très bon concept, une mécanique satisfaisante, mais dont le thème semble hors propos et pourrait améliorer drastiquement la mécanique principale si plus judicieusement choisi. Mais cette méthode prend beaucoup de temps, et est risquée puisqu'à tout moment je pourrais me retrouver plus absorbé par le jeu lui-même que par la recherche de concept novateur. C'est pourquoi je n'ai appliqué cette méthode ci que lors de ma deuxième semaine de stage.

Enfin, la dernière technique, qui pour moi s'est montrée la plus efficace de très loin, a été l'adaptation de jeu flash. Les jeux flash sont un autre vaste sous domaine du jeu vidéo. Ce sont des jeux très bas budget, parfois même sans aucun budget, qui ne sont disponible que sur internet. Ils fonctionnaient à l'époque grâce à Adobe Flash Player, d'où leur nom, et sont codés avec des langages web. Ce sont des jeux très simples, très court, mais qui brillent par leur originalité et leur fluidité. Beaucoup d'entre eux sont des jeux de sport ou de tir, les rendant diamétralement opposés à l'hypercasual. Mais une sous-catégorie des jeux flash est particulièrement plaisante et facile à adapter à l'hypercasual : les jeux de puzzle. Ce sont des jeux basés sur la réflexion, utilisant des mécaniques intuitives et profondes. Il suffit de rajouter la fameuse touche ASMR et d'ajuster le gameplay pour qu'il soit le moins stressant possible, et on obtient un jeu parfaitement hypercasual. L'écrasante majorité des concepts que j'ai pu proposer à Blue Monkey Studio était des jeux de puzzle flash adaptés, et bien que tous n'ont pas fait l'unanimité, certains ont su retenir l'attention de mes collègues, et parfois aussi de l'éditeur.

Et c'est ainsi qu'au bout de cette semaine d'idéation, nous nous sommes retrouvé avec tous les membres de l'équipe afin de sélectionner les meilleures idées à présenter à Voodoo. L'une des miennes a été retenue, celle intitulée Frog Run. J'ai alors dû créer un moodboard, une fiche sur laquelle devait figurer les mécaniques principales du jeu, son ambiance et son thème ainsi que diverses références et inspirations. Ce moodboard, accompagné d'un petit questionnaire recensant des informations générales sur le studio de développement et le jeu imaginé, a donc été envoyé à Voodoo pour présenter Frog Run au concours d'idéation.



Fig 1 :Le moodboard présenté à Voodoo pour décrire l'idée de Frog Run

3.3. Développement d'une solution de gestion des applications automatique

En attendant les résultats du concours, j'ai été affecté à ma première tâche impliquant du développement. Pas du développement de jeu pour le moment, mais le développement d'une solution de Build automatisée. Une fois un prototype de jeu créé sur Unity, il faut exporter le jeu afin d'en créer une application exécutable par n'importe quel smartphone. Les systèmes d'exploitation les plus utilisés par les smartphones et autres appareils mobiles sont Android et iOS. Cependant, les builds que l'entreprise cherche à automatiser sont des builds iOS, puisque les prototypes sont testés sur un public américain, qui possède donc majoritairement des iPhone fonctionnant sous iOS. Donc dans un premier temps je n'ai travaillé que sur les builds iOS. Le but était de faire gagner du temps à l'entreprise en développant une solution permettant de construire automatiquement un exécutable des prototypes de jeu lorsque du code est poussé sur la branche main d'un répertoire Github. La solution que nous avons alors choisie pour faire nos premiers essais est **Github Actions**. Github Actions est un ensemble de fonctionnalités capable d'automatiser diverses tâches relatives aux projets Github, notamment la réalisation de tests unitaires, la création de builds et leur déploiement sur diverses plateformes. Github actions permet aussi la gestion des branches, des erreurs de merge, et la création d'événements, comme un envoi de signal à un fichier de code en cas de push sur une branche spécifique.

Dans un premier temps, j'ai mis en place un workflow. C'est un fichier de code contenant les données des API à invoquer et les commandes de ces API, ainsi que les événements Github qui vont déclencher ces commandes. Ce premier workflow nous a permis de développer des builds pour Android, iOS et Windows Standalone. Bien que les builds iOS sont les seules dont l'entreprise a besoin, les autres builds m'ont été utiles pour faire des tests sur l'efficacité du workflow, n'ayant pas d'appareil compatible iOS à disposition.

Ensuite, j'ai implémenté dans ce workflow plusieurs autres fonctionnalités, comme le stockage des builds sur une branche spécifique de Github, l'intégration de l'API **Diawi** pour permettre de distribuer facilement la build construite, et l'implémentation de l'API **Slack** pour envoyer un message sur un canal spécifique sur le serveur Slack de l'entreprise. Ce dernier point n'a cependant jamais fonctionné correctement.

La principale difficulté que j'ai eu avec Github Actions, outre l'impossibilité de tester les builds iOS, a été la gestion des artefacts. Lorsque Github Actions exécute un workflow, le temps d'exécution est conservé, et les fichiers en sortie sont stockés en tant qu'artefact. Dans sa version gratuite, Github Actions ne permet qu'un certain temps total d'exécution mensuel et un certain volume d'artefacts gardés en mémoire. Bien que les builds Android étaient particulièrement longues à être créées, le temps d'exécution total n'a jamais été problématique. En revanche, le volume des artefacts l'a été. Les builds iOS sont particulièrement plus lourdes que les autres, seuls deux builds suffisent à remplir le quota mensuel. Il existe une fonctionnalité de Github Actions permettant de supprimer ces artefacts, que ce soit à la main ou automatiquement à la création d'une nouvelle build par le workflow. Cependant, les serveurs Github ne se mettent pas immédiatement à jour, ce qui a énormément contraint mes tests sur le workflow.

3.4. Développement du premier prototype (Frog Run)

Alors que mon travail sur Github Actions n'avait pas encore été achevé, le projet a été interrompu à la suite d'un message reçu par Blue Monkey Studio de la part de l'éditeur Voodoo. L'idée du jeu Frog Run présenté au concours d'idéation a remporté la troisième place. L'entreprise a donc décidé de mettre le projet Github Actions en pause et de m'affecter à plein temps au développement du prototype de Frog Run, afin de pouvoir le présenter au second concours de Voodoo.

Dans Frog Run, le joueur contrôle une grenouille qui se déplace dans un environnement en trois dimensions. Cet environnement est composé de murs constituant un labyrinthe et d'obstacles entravant la progression de la grenouille. La grenouille se déplace d'une façon particulière, grâce à sa langue, qui s'accroche à la position où le joueur a touché l'écran. Cette langue fait office de grappin, en plus d'être collante et élastique. Le but du jeu est de trouver la sortie du labyrinthe de chaque niveau, indiqué par une ligne d'arrivée, et de la traverser avec la grenouille. Chaque niveau possède un certain nombre de mouches, qui restent collées à la langue de la grenouille au contact. Et lorsque la grenouille rétracte sa langue alors qu'une mouche y est emprisonnée, la grenouille se met à grossir, induisant plusieurs changements dans le comportement physique du batracien. La grenouille tire plus sur sa langue, est attirée plus fort encore par le point d'ancrage de sa langue, et applique plus de force lors de sa collision avec des obstacles.

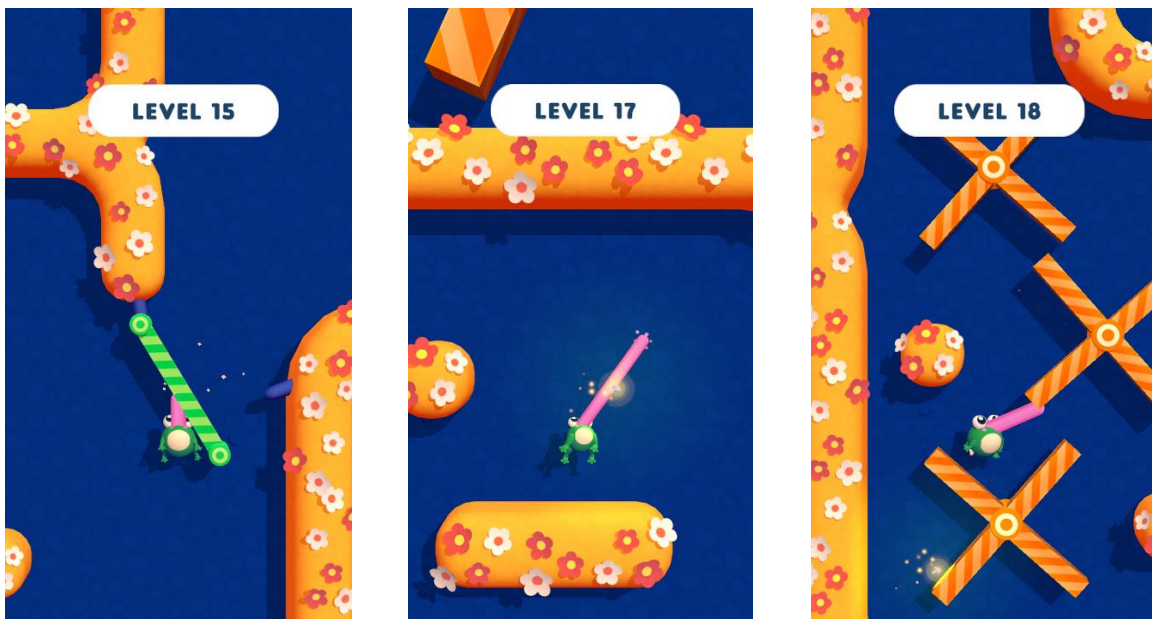


Fig 2 : Images tirées du jeu Frog Run

Sans le savoir, j'ai été assigné à la réalisation de ce qui est probablement le projet le plus difficile que je n'ai jamais développé, surtout que c'est le seul projet sur lequel j'étais le seul développeur. Pour un premier jet dans le domaine de l'hypercasual, le projet s'est avéré extrêmement complexe, notamment au niveau de la physique. Je reviendrai sur les difficultés tout au long de la description du travail accompli.

La première étape du développement est l'étude du terrain. Je veux dire par là analyser comment d'autres développeurs avant nous ont abordé la conception de certaines mécaniques,

comme la langue élastique par exemple. J'ai recherché également sur l'[Unity Asset Store](#) si certaines mécaniques n'avaient pas déjà été développées et mises à disposition des studios, afin de les récupérer pour gagner du temps de développement. Sur ce projet spécifiquement, aucun code de l'Unity Asset Store n'a pu être utilisé, en raison du côté non ouvert à l'extension du code en question. Par exemple, nous avons intégré un asset permettant de créer des cordes qui se plient le long des obstacles rencontrés, mais impossible de rendre cette corde élastique. Et quand nous avons testé un asset de corde élastique, les collisions étaient mal gérées et impossible de rendre la corde collante.

La deuxième étape du prototypage est la réalisation d'assets simpliste. Le but de ces assets est de simuler de manière extrêmement simpliste l'aspect visuel qu'arborera le jeu dans sa version finale. Ils permettent de créer un niveau bac-à-sable utile pour tester toutes les fonctionnalités développées, en attendant que le designer réalise des modèles plus détaillés et esthétiquement appréciables. Ainsi, la grenouille se résumait à une simple sphère verte, sa langue à une ligne rose, et les murs et obstacles à des assemblages de cubes colorés.

Enfin, le développement de code a pu débuter. Les premiers aspects que j'ai travaillés sont les mécaniques principales. D'abord, gérer les inputs de l'utilisateur pour que la langue s'accroche au point de contact du doigt avec l'écran. Puis faire grossir progressivement la grenouille au contact d'une mouche, et gérer les collisions entre la grenouille et les murs et autres obstacles.

Vint ensuite les bases de la physique de la grenouille. D'abord, l'ajout d'un léger effet gelée remuante sur la grenouille quand elle percute un obstacle solide, ainsi qu'un effet rebondissant. Cette amélioration physique vise à donner de la matière à la grenouille, pour lui donner de la cohérence dans son univers, et aussi pour inciter le joueur à faire se percuter la grenouille dans tous les sens, car la torture animale ce n'est pas bien mais dans un jeu vidéo cartoonesque c'est drôle et satisfaisant. Le point physique suivant, et pas le moindre, est l'ajout de l'effet grappin élastique à la langue de la grenouille afin qu'elle soit attirée vers son point d'ancrage, et qu'elle puisse se déplacer de la manière voulue par les mécaniques du jeu. En parallèle, puisque cela concerne aussi la langue de la grenouille, j'ai développé les aspects collant et matériel mou de la langue. Il fallait donc faire de la détection de collision sur une langue en mouvement permanent, dont la forme change, dans un environnement lui-même changeant, afin de la plier selon les obstacles et de se coller à d'autres. Ces deux points spécifiquement ont été parmi les plus retards, puisque la grenouille se déplaçant dans un environnement en trois dimensions mais uniquement selon un plan, on pourrait être tenté d'intégrer de la physique 2D pour ses déplacements. Or, les `springJoint2D`, ou les ressorts en deux dimensions intégrées au moteur Unity, ne sont pas adaptés pour obtenir le comportement recherché, contrairement au `springJoint` version 3D. Donc une physique 3D serait plus appréciable pour gérer les déplacements de la grenouille. Or, au contraire, la physique en deux dimensions est bien plus performante pour gérer de la collision point par point, très utile pour implémenter efficacement le comportement de la langue. Et malheureusement, Unity ne tolère pas qu'un objet soit géré selon une logique en deux dimensions et en trois dimensions simultanément. J'ai donc dû jongler entre les deux physiques et trouver une solution de contournement pour conserver les aspects intéressants de chacune sur un seul objet. Mais je suis quand même parvenu à un résultat plus que satisfaisant, les mouvements de balanciers de la grenouille étant hypnotisants et le déplacement assez jouissif. A ce niveau-là du développement, je n'en avais pas du tout terminé avec ces deux aspects de la physique du jeu, puisqu'au fur et à mesure que de nouvelles fonctionnalités ont été implémentées, de nouveaux bugs liés à ces comportements sont apparus, et ne pouvaient pas être négligés étant donné qu'ils entravaient complètement la progression du joueur.

Avec des assets de bases, des mécaniques basiques et de la physique encore peu convaincante, le prototype commençait à ressembler à un jeu. Mais le but d'un prototype de jeu hypercasual est de

proposer une version miniature du jeu final, avec un design déjà avancé, et des mécaniques théoriquement très proches de la version finale. Je me suis donc attelé à l'amélioration de la physique détail par détail, notamment sur le comportement collant de la langue et de l'impact de l'environnement sur celle-ci, et à divers visuels, notamment l'ajout d'un effet zigzag lorsque la langue est déployée. Mon collègue designer ayant terminé une première version des modèles du jeu, j'ai pu les ajouter au projet et travailler sur l'animation de ceux-ci (bien qu'il ne s'agît pas tout à fait d'une animation au sens classique du terme sur Unity mais d'un comportement physique). En particulier, j'ai ajouté un effet ragdoll aux membres de la grenouille pour accentuer l'effet torture.

La physique et l'aspect visuel ayant été améliorés, je suis passé au développement des obstacles. Il y a en tout quatre types d'obstacles dans le prototype :

- Les murs destructibles, reconnaissables par leur visuel semblable à un damier. Ils peuvent être traversés par la grenouille si celle-ci est assez lourde, elle doit donc avoir mangé un certain nombre de mouches au préalable.
- Les plateformes accrochées, ce sont des barres rayées attachées à deux murs, et qui se détachent si la grenouille y colle sa langue alors qu'elle est suffisamment lourde.
- Les moulins, qui sont des plateformes en rotation autour de leur centre de gravité, qui ont pour but d'entraver la progression de la grenouille.
- Les barres mouvantes, qui effectuent des translations d'un point A vers un point B et vice versa. En plus d'entraver la progression de la grenouille, elles peuvent aussi forcer le joueur à accomplir certaines actions avec un certain timing ou aider la grenouille à prendre de l'élan avant de s'élancer dans une certaine direction.

L'avant dernière étape du développement du prototype de Frog Run a été la création de niveaux. Les prototypes classiques d'hypercasual possèdent une dizaine de niveaux, avec une difficulté minutieusement graduée, et dont les deux premiers servent de tutoriel. Les niveaux durent rarement plus d'une minute trente à être accomplis. Ce schéma général est celui que nous avons choisi pour le développement des niveaux de Frog Run. Le premier niveau sert de tutoriel pour que le joueur puisse expérimenter librement le système de déplacement grâce à la langue. Le deuxième introduit les mouches, qui entre temps sont devenues des lucioles par soucis d'esthétisme, et la mécanique de grossissement. Le troisième introduit les premiers obstacles, et un nouvel obstacle est ensuite ajouté tous les deux niveaux. Une fois les niveaux agencés et implémentés au projet, s'en est suivie une phase de tests pour régler tous les bugs que l'équipe a pu expérimenter. Suite à cette phase de tests, l'effet zigzag de la langue a été supprimé car pouvoir gérer la physique de la langue au moment du zigzag de manière à ne pas générer de bug aurait entraîné du temps supplémentaire de développement, et Frog Run aurait alors demandé bien trop de temps de développement pour un prototype.

La dernière étape, qui est toujours la dernière étape quel que soit le prototype de jeu développé, est le polishing. Le principe du polishing est d'effectuer des ajustements sur tous les aspects du prototype. Par exemple, l'ajout de divers systèmes de particules lorsque la grenouille entre en collision avec des obstacles, et l'ajustement de la forme et la couleur de ces particules, ou encore l'adaptation du recul de la caméra quand la grenouille grossit. L'objectif est de rendre le jeu plus fluide, plus plaisant et satisfaisant à jouer et contempler.

Et ceci fait, le prototype de Frog Run a officiellement été terminé. Malheureusement, puisque j'étais seul sur le projet, et que les mécaniques à développer étaient particulièrement chronophages

par leur complexité, je n'ai pas pu terminer le prototype avant la date buttoir imposée par Voodoo pour son second concours.

Possédant dorénavant une version complète du prototype de FrogRun, il ne manque plus qu'à poster le prototype sur les stores. Encore une fois, cette phase se fait en plusieurs étapes, mais cette fois les étapes se font en parallèle et non plus successivement.

La première chose à faire est d'ajouter au prototype la sdk Voodoo. Cette sdk fournie par l'éditeur permet de faire de la collection de statistiques sur le jeu. Ces statistiques permettent de savoir si le jeu sera rentable dans l'éventualité de la publication d'une version complète, et si ce n'est pas le cas, identifier le ou les points bloquants. Si ces points bloquants peuvent être outrepassés, le studio de développement peut choisir avec l'accord de l'éditeur de faire faire une itération, c'est-à-dire une version plus améliorée du prototype. Dans l'idéal, l'itération ne change qu'un seul point du jeu (une mécanique, le design général, le thème...), mais elle peut changer bien plus qu'un seul point. Ces statistiques sont la rétention (le temps moyen d'un joueur sur le jeu), le nombre de fois que chaque niveau a été terminé par l'ensemble des joueurs (pour identifier un niveau trop difficile, ou qui provoque une perte d'intérêt), le nombre de fois que les joueurs relanceront le jeu, sur combien de jours le jeu sera relancé, ou encore combien le jeu coûterait (Le très important CPI, coût par installation).

La sdk intégrée, on peut générer une build ios du prototype, et créer un bundle sur le site de Voodoo pour lier le prototype à un bundle sur App Store Connect et sur Facebook Developer, et de récupérer les statistiques que ce prototype a obtenu au bout d'une semaine.

Facebook Developer permet de gérer la publicité autour du jeu. Le studio doit créer des vidéos appelées **vidéos CPI**. Ces courtes vidéos ont pour but de montrer le prototype sous son plus bel aspect, en mettant en avant le plus vite possible ce que le jeu propose de plus attrayant. Comme il s'agit de publicité, elles ne reflètent pas forcément le jeu exactement tel qu'il est. Par exemple, sur l'une des vidéos CPI de Frog Run, la grenouille a été remplacée par un bonhomme sous conseil de l'éditeur, car voir un bonhomme s'agiter dans tous les sens en ragdoll pourrait potentiellement être plus satisfaisant que son équivalent avec une grenouille, et permettrait donc à la publicité d'être plus efficace. Ces vidéos sont ensuite diffusées sur Facebook dans les localisations où le prototype sera disponible.

App Store Connect permet de créer la page du jeu sur le magasin App Store. Des informations sont requises pour créer cette page, comme la localisation de diffusion, des informations relatives à l'entreprise à l'origine du jeu, ou des captures d'écran du jeu sous différents formats (iPhone, tablette...).

Une fois tous ces bundles créés et le lien entre eux établi, il n'y a plus qu'à attendre une semaine pour consulter les statistiques du jeu et agir en fonction des résultats.

3.5. Développement du deuxième prototype (Flip and Snap)

En attendant les résultats de Frog Run, j'ai été affecté au développement d'un autre jeu imaginé cette fois par l'un de mes collègues. Il s'agit d'un jeu appelé Flip and Snap (ou Flip n'Snap, ou Flip & Snap). C'est un jeu de type puzzle se déroulant sur un plateau quadrillé. On contrôle une pièce colorée en trois dimensions ayant une taille très spécifique, par exemple la pièce peut mesurer une case de hauteur, une case de largeur et deux cases de longueur. En glissant son doigt dans une direction, on fait pivoter la pièce dans cette même direction. Cependant, la pièce ne peut pas pivoter si les limites du quadrillage ou une autre pièce se trouve à l'emplacement d'arrivée de celle-ci. Chaque pièce possède un emplacement spécial sur lequel elle doit se positionner. Une fois une pièce positionnée, une autre tombe sur la grille et doit à son tour être placée correctement. Le but du jeu est de placer toutes les pièces du niveau pour former un dessin en **pixelart**. En plus du designer, nous étions cette fois deux développeurs sur ce projet, mon tuteur en entreprise et moi.

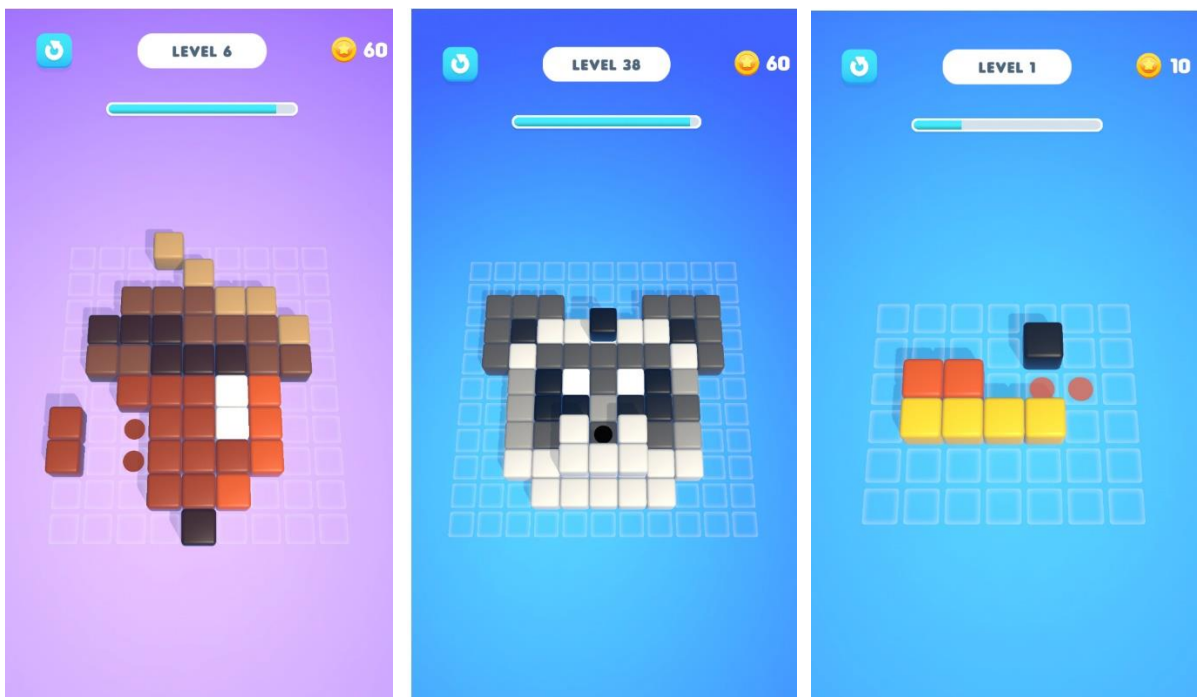


Fig 3 : Images tirées du jeu Flip and Snap

Contrairement à Frog Run, la simplicité des mécaniques du jeu ne nécessite pas la recherche de scripts déjà existant sur l'Unity Asset Store. Mais à l'instar de Frog Run, le projet a débuté par la création d'assets très basiques, les grille et les pièces étant intégralement constituées de cubes.

Une fois les assets en place, j'ai développé les mécaniques principales du jeu, à savoir la chute d'une pièce sur le terrain, son déplacement selon les inputs du joueur et selon les autres pièces, la création de la grille, et la notion d'emplacement pour chaque pièce.

Je me suis ensuite lancé dans une grosse phase de level design. Dans un premier temps, j'ai dessiné sur le logiciel Photofiltre (une préférence personnelle, pas une suggestion de la part de l'entreprise) des dessins en pixelart très minimalistes (entre 15 x 15 et 20 x 20 pixels). Puis j'ai intégré ces dessins dans le jeu pour en faire un niveau, en situant les emplacements de chaque pièce à leur position sur le dessin, et en déplaçant la pièce elle-même sur la grille en effectuant des déplacements

depuis l'emplacement de la pièce pour satisfaire la faisabilité du niveau. Dès la phase de dessin, des nombreuses difficultés se sont présentées :

- Tout d'abord, le dessin doit-être divisible en plusieurs pièces, mais puisqu'une pièce constitue une partie du dessin, il faut que l'agencement de ses couleurs soit le même quelle que soit l'orientation de la pièce. Ce qui signifie qu'on ne peut pas avoir une pièce de format 2 x 1 x 1, composée donc de deux pixels, avec deux couleurs différentes. Chaque pièce doit donc disposer d'une symétrie dans l'agencement de ses couleurs.
- Réussir à faire un dessin en 10x10 pixel est particulièrement difficile si on veut que l'objet ayant servi de modèle au pixelart soit facilement identifiable par tout le monde. Au contraire, une grille 20x20 est particulièrement grande, et pose des problèmes de précision et de lisibilité sur les petits écrans de téléphone. Une solution envisagée à été de diviser chaque cube composant une pièce en huit pixels (quatre par face), pour ainsi avoir des dessins plus complexes sur de plus petites grilles, mais malheureusement cela rendait le premier point encore plus difficile à réaliser.
- Pour pallier le problème de symétrie des couleurs, nous avons pensé à créer des pièces asymétriques, comme une pièce coudée (une pièce 2x2x1 à laquelle il manquerait une case). Cependant, après avoir testé un niveau réalisé ainsi, nous nous sommes rendu compte que la difficulté du jeu devenait abyssale avec de telles pièces.

La première version du prototype a été réalisée sur des grille 10 x 10 cases, avec des pièces asymétriques composées de 8 pixels par morceau de pièce. Après seulement deux niveaux réalisés ainsi, la difficulté du jeu nous a dissuadé de continuer dans cette direction. Donc la seule solution possible pour réaliser un bon jeu à partir de ce concept a été de réaliser des niveaux sur des grilles au maximum de format 15 x 15, avec des pièces uniquement symétriques et pas trop grandes, et donc des couleurs symétriques sur chaque pièce, ce qui contraint énormément le pixelart.



Fig 4 :Pixelart typique de la première version de Flip and Snap

Avec cette nouvelle version, nous avons réalisé un nouveau problème, les pièces étant petites, le pixelart est divisé en beaucoup d'entre elles, en moyenne quatorze pièces par dessin. Or, le placement des pièces à leur emplacement prend entre dix et trente secondes en fonction du niveau du joueur. Ce qui donne dans le pire des cas sept minutes de temps de jeu par niveau, plus de deux minutes dans le meilleur des cas. Ce temps de jeu par niveau est malheureusement bien trop élevé. Nous avons donc pour la dernière version du prototype gardé la possibilité de jouer avec de grandes pièces pour réduire drastiquement la durée de résolution d'un niveau. Pour certains niveaux, nous avons aussi ajouté la notion d'obstacles, qui sont des pièces placées dès le début du niveau sur la grille.

Ces obstacles ont trois avantages majeurs : ils permettent de créer des pixelarts pas forcément composé uniquement de pièces symétriques, le nombre de pièces à placer est réduit (et donc la durée des niveaux aussi), et enfin cela rajoute du défi dès le début du niveau car il y a déjà des obstacles qui entravent le déplacement de la première pièce.

Une fois les mécaniques intégralement développées et les niveaux créés, nous avons implémenté le design du jeu. Contrairement à Frog Run, le design supplémentaire de ce jeu n'a pas nécessité l'ajout de rendu physique et a donc été bien plus simple. Les designs ajoutés ont été un fond coloré avec un gradient, une grille esthétique, des matériaux spéciaux pour les pièces afin qu'elles semblent avoir une texture satisfaisante, et des éclairages particuliers (des shaders).

L'ultime difficulté sur ce projet a été la dernière fonctionnalité que nous souhaitions ajouter au jeu. Le jeu dans son état manquait cruellement d'un petit quelque chose. Il nous semblait un peu vide, pas très vif et donc pas très satisfaisant. Mon tuteur en entreprise a eu l'idée de donner un comportement physique particulier aux pièces lorsqu'elles tombent ou lorsqu'elles pivotent. Dans un premier temps, il a pensé à faire des pièces en pierre, très rigides donc, pour qu'elles produisent un bruit satisfaisant en entrant en mouvement. Cependant, l'aspect sonore ne suffit pas à rendre le jeu plus palpitant. Alors sa seconde idée a été de donner une texture de slime aux pièces. Celles-ci devaient gigoter comme un flan à chaque collision ou déplacement, pour créer un effet satisfaisant garanti. Pour ajouter cet effet physique, trois possibilités s'offraient à nous : utiliser un shader, coder manuellement le déplacement des **mesh** de la pièce ou créer une animation sur Unity. Personne ne sachant comment coder de shader rapidement ex-nihilo, nous avons cherché des shaders sur l'Unity Asset Store, mais aucun de ceux testés n'a fourni un résultat qui nous convenait. J'ai de mon côté développé le déplacement des mesh via script, quand mon tuteur a créé des animations. Finalement, ses animations ont été retenues, car mon script générait quelques bugs liés aux collisions de la pièce.

C'est donc une version de Flip and Snap constituée de dix niveaux qui a été envoyée en test, de la même manière que Frog Run.

3.6. Développement du troisième prototype (Shape Together)

Lorsque nous avons terminé Flip and Snap, deux autres jeux étaient alors en cours de développement. Mon tuteur en entreprise a rejoint l'un de ces deux projets, et moi l'autre. Un seul autre développeur était alors affecté sur ce projet. Son nom est Shape Together. Dans ce jeu, On doit reconstituer un ou plusieurs objets qui sont décomposés en plusieurs pièces. Par exemple, une maison peut être décomposée en son toit d'un côté, et ses murs de l'autre et le but est de replacer le toit au-dessus des murs ou les murs en dessous du toit. Cependant, ce déplacement se fait selon des contraintes. Premièrement, toutes les pièces sont placées sur des rails. Elles ne peuvent coulisser que le long de ces rails lorsque le joueur fait glisser son doigt depuis la pièce vers l'une des extrémités du rail. Ensuite, les pièces possèdent des collisions entre elles, et se bloquent si elles se situent sur un même rail ou à l'intersection entre deux rails.

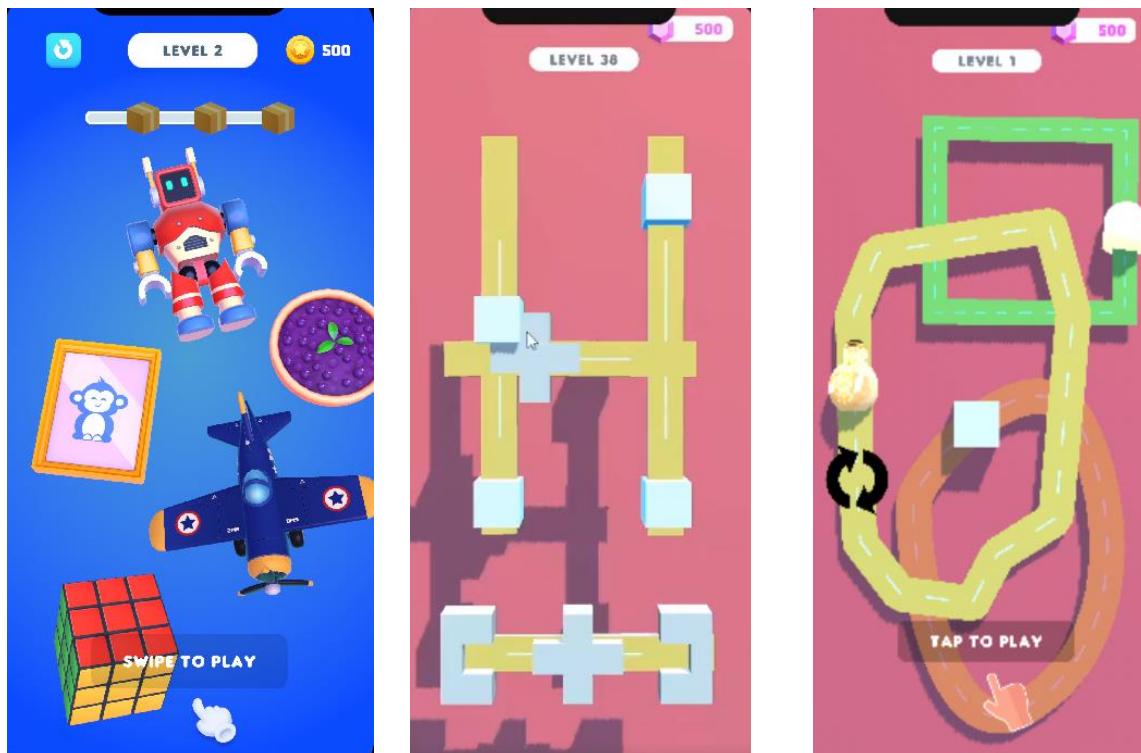


Fig 5 : Images tirées du jeu Shape Together

Mon collègue sur ce projet avait déjà bien entamé le développement de la mécanique des rails et du coulisement des pièces le long de ces rails, son objectif désormais est de gérer les collisions entre les pièces qui étaient alors hasardeuses, de régler les quelques bugs majeurs présents et d'ajouter un effet lorsqu'une pièce est bien placée par rapport à une autre pour que les deux pièces semblent se clipser. Pendant ce temps, j'avais la charge d'imaginer et d'implémenter le level design. D'abord, j'ai dessiné une panoplie de niveaux sur papier, puis sur Photofiltre pour plus de lisibilité. Ces niveaux comprenaient un certain nombre de pièces par objet, un certain nombre d'objet, un certain nombre de rails avec un agencement particulier. Nous avons ensuite sélectionné quels dix niveaux nous allons garder, et choisi leur ordre en fonction de la courbe de difficulté de chaque niveau et des nouveaux ajouts par rapport aux niveaux précédents. Certains niveaux ont été réadaptés car impossible à compléter dans certains cas, d'autres ont été malgré tout conservés pour faire office de niveau bonus, car contrairement à Flip and Snap, les niveaux étaient théoriquement bien plus rapides à compléter (moins d'une minute pour les niveaux les plus difficiles).

Une fois que mon collègue a implémenté la mécanique permettant de clipser les pièces entre elles, nous nous sommes rendu compte d'un problème majeur qui a rendu tout mon travail sur le level design obsolète. La nouvelle mécanique veut que lorsqu'on déplace une pièce A vers une pièce B, si A doit se clipser à B alors la pièce B est déplacée vers sa position exacte par rapport à A, et les deux pièces ne forment alors plus qu'une seule pièce située sur le rail de A. Je vais illustrer ce problème avec un exemple. Imaginons un niveau constitué de trois rails, deux horizontaux parallèles entre eux (appelons les A et B) et un vertical qui coupe les deux autres en leur centre (appelons le C). Soit un jouet robot, notre objet dans ce niveau, découpé en trois parties, la tête le torse et les jambes, tel que la tête est sur le rail A, le torse sur le rail C et les jambes sur le rail B. Une manière de résoudre ce niveau est de déplacer la tête à l'intersection entre les rails A et C, et d'amener le torse du robot sous la tête. On obtient ainsi une nouvelle pièce tête/torse sur le rail C, puis on place les jambes à l'intersection entre les rails B et C, et on descend l'ensemble tête/torse pour le clipser aux jambes, et on termine le niveau avec un robot intégralement reconstitué. Cependant, si on recommence le

niveau, mais en commençant cette fois par déplacer le torse à l'intersection entre les rails A et C, puis ramener la tête vers le torse, alors on obtient un ensemble tête/torse comme tout à l'heure, mais cette fois-ci sur le rail A. Or, les jambes se trouvent sur le rail B, et les rails A et B ne s'intersectent pas puisqu'ils sont parallèles. Le joueur est donc bloqué.

Face à ce problème, trois solutions s'offraient à nous :

- soit on admet que le joueur peut être bloqué lors d'un niveau, et doit le recommencer le niveau du début via un bouton reset. Mais cette solution n'est de notre point de vue pas viable, car recommencer un niveau du début peut créer de la frustration chez le joueur, et la frustration est la pire sensation qu'un joueur puisse avoir face à un jeu hypercasual.

- soit on donne un ordre de priorité aux rails, pour que les pièces des rails le plus prioritaires attirent sur leur rail les autres pièces en cas de bon placement. Ainsi dans l'exemple précédent, si le rail C est prioritaire par rapport à A et B, que l'on déplace la tête vers le torse ou le torse vers la tête, c'est le torse qui va attirer la tête sur son rail pour la clipser. Le niveau ne sera donc jamais bloqué. Cette solution, bien qu'alléchante, possède néanmoins un souci majeur : selon mon collègue, la façon dont les rails et le clipsage sont implémentés ne permettraient pas d'ajouter un système de priorité facilement. Cette solution a donc elle aussi été abandonnée.

- soit on crée des boucles de rails. C'est-à-dire par exemple qu'on relie entre elles les extrémités des rails A et B pour ne former plus qu'un seul rail de forme rectangulaire tel que les pièces situées dessus puissent tourner tout autour de ce rail sans jamais être bloquée.

Par défaut donc, cette dernière solution a été retenue, et j'ai été chargé de retravailler tous mes niveaux pour ajouter des boucles aux endroits où le niveau pouvait être bloquant à cause de la nouvelle mécanique.

Ceci fait, une réunion a été organisée par les fondateurs de l'entreprise pour faire un point sur le projet. Les niveaux étaient tous prêts, mais mon collègue n'arrivait pas à corriger certains bugs majeurs du jeu. Il s'est avéré que la cause de ces bugs était l'ensemble de scripts qui provenait de l'Unity Asset Store qui a servi à faire fonctionner le système de rail. Pour corriger cela, il aurait fallu recommencer le code du projet depuis le début, puisque ces scripts sont la pierre angulaire du projet. Mais le développement du projet ayant déjà duré plusieurs semaines de plus que celui d'un projet classique, nous avons décidé de le mettre en suspend le temps de constituer une base de code réutilisable et extensible capable de gérer les rails et les collisions sans générer de bugs. En attendant, nous avons été dispersés sur divers autres projets. A l'heure où j'écris ce rapport, le projet Shape Together a été repris par mon collègue après plusieurs mois.

3.7. [La première itération sur un prototype \(Flip and Snap\)](#)

De mon côté, j'ai été affecté au développement d'un autre jeu. Mais cette fois-ci, il ne s'agissait pas d'un nouveau jeu, mais au contraire, d'un jeu que je connaissais très bien. Après l'obtention de résultats plutôt encourageants, notre éditeur nous a enjoint à développer une itération de Flip and Snap.

Ici, il ne s'agit plus de développer un prototype ex-nihilo, ni même sur la base de quelques scripts réutilisés. L'itération consiste en l'amélioration d'un ou plusieurs points du jeu, et en un prototype plus long que la première version. Et la première étape de l'itération est l'analyse des résultats, afin

d'identifier les points à améliorer. Je parlerai plus en détail de l'analyse des résultats dans la section éponyme de ce rapport. Il en est ressorti que les niveaux étaient trop difficiles et que le jeu manquait encore d'objectif sur le long terme, puisque finir tous les niveaux ne suffisait pas à lui seul.

Nous avons donc ensemble, mon tuteur en entreprise, le designer et moi-même, ajouter plusieurs fonctionnalités majeures au jeu.

La première de ces fonctionnalités est la création d'un nombre significatif de niveaux. Nous sommes passé d'une version de dix niveaux à une version de presque cinquante niveaux. L'idée était de réunir ces niveaux par groupes de cinq au maximum en thèmes. Lorsque le joueur débute au jeu, il possède le thème « salle de bains ». Il va donc devoir reconstituer un canard en plastique dans le premier niveau, un savon dans le suivant etc... Puis une fois tous les dessins du thème achevés, il débloquent un tout nouveau thème. Ce système de thème permet de donner encore plus envie au joueur de continuer à jouer au jeu, puisqu'à la fin de chaque niveau, il voit sa progression avant de pouvoir débloquent le prochain thème, et qu'il est curieux de découvrir de quel thème il s'agit, ainsi que de savoir quels sont les dessins qui le compose. Nous avons de cette manière créé dix thèmes pour le jeu. Mon rôle exact dans cette partie a été d'imaginer les thèmes, de créer les dessins en pixelart des tous les nouveaux niveaux et de réadapter les anciens en fonction des thèmes choisis, et de choisir le placement des pièces quand elles tombent sur le plateau. A ce propos, une mise au point a été faite sur le placement des pièces. Le nombre de déplacements nécessaire pour déplacer une pièce vers son emplacement final dépend du thème dans lequel on se trouve afin d'adapter la difficulté. Ainsi, deux déplacements suffisent pour les premiers thèmes, mais entre six et huit pourront être nécessaires au cours des derniers. Une fois les niveaux conçus, il a fallu les implémenter dans le prototype. Nous avons tout trois ajouté les niveaux en se répartissant les thèmes.

L'autre fonctionnalité majeure a été l'ajout d'une troisième dimension au jeu. Jusqu'ici, les pièces en trois dimensions permettaient de constituer un dessin en deux dimensions seulement. Nous avons eu l'idée de créer des niveaux sur plusieurs couches pour rajouter de la profondeur au jeu (c'est le cas de le dire). Chaque thème possède un tel niveau, son dernier niveau. L'objectif est de remplir dans un premier temps la couche du bas (par exemple la tête d'un chien), et une fois cette couche remplie, étonnamment, le niveau n'est pas fini et propose de remplir une seconde couche (pour poursuivre l'exemple, le museau du chien). Ce genre de niveau, en plus d'ajouter de la diversité dans les niveaux, permet d'accrocher encore plus le joueur au jeu, et donc d'obtenir une meilleure rétention.

Bien que ce n'est pas une fonctionnalité, la refonte graphique du jeu et de son interface utilisateur mérite d'être mentionnée dans les changements majeurs, bien que je n'y ai pas directement participé.

Parmi les changements mineurs, on peut citer l'ajout des obstacles en début de partie. Ce n'était qu'une idée présente sur certains niveaux lors de la première version du prototype, les obstacles sont devenus monnaie courante sur nos niveaux dans cette itération, en raison de la diminution de temps de jeu nécessaire puisque, autre détail, nous avons décidé de limiter les pièces déplaçables à de très petites pièces pour les premiers thèmes (2 x 1 ou 3 x 1), et des pièces à peine plus grandes pour les derniers thèmes (4 x 1, 2 x 2 ou parfois 4 x 2). Et puisque les pièces sont plus petites, il est nécessaire d'en déplacer plus pour reconstituer un dessin incomplet, au risque que le joueur devine ce que représente le premier dessin avant même d'avoir placé la première pièce, ce qui ruine totalement l'effet qui apporte le plus de satisfaction dans ce jeu.

Un autre changement mineur est la possibilité d'utiliser les pièces obtenues à la fin de chaque niveau pour placer correctement la pièce courante. L'option de placement automatique se dévoile une fois que le joueur a suffisamment déplacé sa pièce sans réussir à la placer correctement, ce qui peut lui éviter d'être bloqué trop longtemps.

La plus grande difficulté de cette itération aura été de réaliser autant de pixelarts sur des thèmes très spécifiques avec une aussi petite résolution de sorte que tous les dessins restent compréhensibles. Cependant, nous avons trouvé une technique pour rendre tout cela possible, à défaut d'être évident. Nos thèmes n'ont pas été sélectionnés au hasard, il s'agissait juste de la compilation des dessins les plus faciles à faire en pixelart. C'est pour cela que trois de nos thèmes correspondent à l'univers animalier (animaux de la ferme, animaux de compagnie et animaux sauvages), étant donné que les animaux sont particulièrement faciles à représenter même en très basse résolution. Ceci est peut-être dû à la culture imagée qui nous entoure, comme les bandes dessinées ou les dessins animés, qui avec l'expérience nous ont permis d'identifier des animaux sur des dessins uniquement sur la base d'éléments très caractéristiques, par exemple des grandes oreilles et une trompe pour un éléphant, mais l'éléphant pourrait aussi bien être vert qu'on reconnaîtrait l'éléphant juste par la présence de ces deux éléments. Un autre thème très facile à réaliser sont les drapeaux, puisque les drapeaux sont généralement rectangulaire et composés de rectangles de diverses couleurs, tandis que le pixelart est composé de carrés qui assemblés forment des rectangles de tous formats.

Une fois ces fonctionnalités développées, mes collègues se sont chargés du polishing du prototype. Mais cette étape ne nécessitant pas la présence d'une troisième personne sur le projet, j'ai été affecté à une tâche que j'avais autrefois laissé en suspens.

3.8. [Retour sur l'automatisation de gestion d'application](#)

J'ai ensuite repris mes travaux sur Github Actions. En tout cas, mes travaux sur l'automatisation de la génération de build, et de toutes les fonctionnalités qui en découlent. Mais cette fois-ci, on m'a conseillé d'étudier **Unity Cloud**. Unity Cloud fonctionne sur le même principe que Github Actions, dans le sens où on peut instaurer des workflows et des événements qui peuvent déclencher ces workflows. La différence est que sur Unity Clouds, il n'y a aucun fichier de code à rédiger, tout est automatisé, il suffit de remplir des champs dans les paramètres associés à un workflow (par exemple la version d'unity, quel système d'exploitation pour la build...) et associés à un événement (le repository Github associé, l'élément déclencheur...). De plus, Unity Clouds permet l'intégration d'API externes encore plus simplement que Github Actions. De plus, les restrictions sur les artefacts sont bien moindres sur Unity Clouds que sur Github Actions, puisque le temps de build est réduit mais toujours conséquent, et l'espace de stockage des artefacts est bien plus élevé. Le seul inconvénient est la non-gratuité du service, mais comme mon entreprise avait déjà effectué le paiement, je n'avais pas à prendre en compte ce critère dans mon comparatif entre les deux solutions. En l'espace d'une semaine, j'ai réussi à générer une build ios via Unity Clouds, générer un lien pour accéder à cette build, et implémenter l'API Slack pour recevoir une notification contenant le lien sur l'un de nos canaux pour être prévenu lorsque la build est terminée afin de la télécharger pour la tester.

L'étape suivante a été de permettre la distribution automatique sur l'App Store de la build ios. Et c'est ici que les choses se sont gâtées. Unity Cloud possède une solution intégrée pour déployer automatiquement une build Android sur le Play Store de Google, et permet même d'automatiser le remplissage de certains champs comme les informations relatives à l'entreprise, l'association de diverses données du jeu aux données de la page du magasin (par exemple une politique de confidentialité), ou encore plus étonnant, les screenshots (captures d'écran) qui permettent sur la page du magasin de donner un aperçu visuel du jeu. Mais rien de tout cela n'est encore possible sur Unity Cloud si on ne souhaite pas publier une build Android sur le Play Store mais une build ios sur l'App Store. Unity Clouds est un service encore trop récent et donc incomplet. Cependant, si Unity

Clouds ne propose pas d'automatiser la publication sur l'App Store, il permet d'exécuter des fichiers de code à la fin de l'exécution des workflows. Le but est de récupérer les données en sortie du workflow, notamment la build ios et divers paramètres qui lui sont associés, et d'utiliser ce code pour déployer cette build sur l'App Store.

Notamment, il est possible de créer un fichier .sh, qui fonctionne via TestFlight, un service en ligne permettant d'effectuer une installation et des tests en direct sur les build ios.

Sinon, on peut créer un webhook en exploitant les fonctionnalités de L'API **Apple Store Connect**. L'API possède l'équivalent de toutes les fonctionnalités citées plus haut valables pour la distribution sur le Play Store.

Cependant, je n'ai pas trouvé comment spécifier le remplissage de certains champs dans les fichiers .sh, comme la politique de confidentialité ou le prix du jeu. Et en ce qui concerne l'API App Store Connect, bien que la documentation soit très claire et fournie, elle manque cruellement d'exemples, et je n'ai pas trouvé de tutoriel expliquant en détail les fonctionnalités que je cherchais à implémenter. Donc j'ai dû suivre mon intuition pour créer le webhook, mais hélas aucun n'a jamais fonctionné sans que je ne sache pourquoi.

Cependant, ce travail, bien que n'ayant pas porté ses fruits, m'a permis de découvrir l'existence d'une autre solution pouvant nous permettre de déployer notre build ios, **Fastlane**. En plus d'être complètement gratuit, Fastlane propose de générer la build ios, d'exploiter diverses API, notamment dans notre cas Slack et Diawi, pour notifier de la réussite de la génération de build, de créer une page sur l'App Store et de remplir ses champs automatiquement, et même de générer les captures d'écran du jeu. La solution fastlane est plus complète que Unity Clouds et Github Actions, sans restriction de temps ni d'espace, bien mieux documentée et bien plus utilisée par les entreprises comme les particuliers. Cela fait de Fastlane la meilleure solution de loin pour parvenir à notre objectif.

Mais bien sûr, il y a un hic. Fastlane est une application à installer sur son ordinateur et non un service en ligne. Or, je dispose d'un système d'exploitation Windows, et Fastlane, bien que partiellement compatible avec Windows, n'est pleinement opérationnel que sur Mac Os. Et ne disposant pas d'un Mac, j'ai tenté de faire sans. Il m'est rapidement apparu que sans possibilité d'effectuer la build ios via Fastlane, je ne pouvais pas aller plus loin.

J'ai organisé une réunion avec les fondateurs de l'entreprise pour leur expliquer mon avancement, et leur décrire mes analyses sur les diverses solutions et les problèmes que j'ai pu rencontrer avec eux jusqu'à tomber dans l'impasse. À la suite de cette réunion, j'ai à nouveau été affecté au développement d'un jeu dont j'avais eu l'idée : Light the Fuse.

3.9. Développement du dernier prototype (Light the Fuse)

Light the Fuse est, à l'instar de Flip and Snap et Shape Together, un jeu de puzzle. Le joueur dispose d'une grille remplie de tuiles carrées. Sur chaque tuile se trouve des cordes qui relient plusieurs arrêtes du carré entre elles. En dessous de la grille se trouve une rangée d'allumettes, une sous chaque case de la première ligne de la grille en partant du bas. Au-dessus se trouvent des fusées de feux d'artifices, une sur chaque case de la première ligne en partant du haut. Certains niveaux proposent des variantes, par exemple en retirant certaines allumettes. Le joueur peut en touchant une tuile la faire tourner de quatre-vingt-dix degrés dans le sens horaire, changeant ainsi l'orientation des cordes sur cette tuile par la même occasion. L'objectif du jeu est de relier les allumettes aux fusées grâce aux cordes. Lorsque

ceci se produit, l'allumette va mettre le feu à la corde juste au-dessus d'elle, et la flamme va se propager aux autres cordes de proche en proche, jusqu'à allumer la ou les fusées qui vont décoller. Les tuiles associées aux cordes consumées sont ensuite détruites, la gravité fait tomber les tuiles alors en suspension, et d'autres tuiles apparaissent au-dessus de la grille pour remplir intégralement celle-ci. Le but de chaque niveau est de réussir à accumuler assez de score pour pouvoir passer au niveau suivant. Chaque niveau permet d'obtenir jusqu'à trois étoiles. Obtenir la première étoile permet de passer le niveau, la deuxième étoile ne consiste qu'en un intermédiaire entre les deux autres étoiles, et la troisième étoile permet au joueur de voir son score exploser exponentiellement. Le système de score est calculé selon trois facteurs :

- Plus le joueur déclenche de fusées simultanément, plus il obtient un score élevé. Le nombre de points de score obtenu suit à peu près la suite de Fibonacci, mais multipliée par cent, puisque les gros chiffres offrent plus de satisfaction, les joueurs préfèrent lorsqu'ils voient leur barre de score exploser. Déclencher une seule fusée rapporte cent points (au lieu de deux cents comme l'exigerait une vraie suite de Fibonacci), deux fusées rapportent trois cents points, trois fusées cinq cents points, quatre fusées huit cents points etc...

- Si le joueur réalise un combo, le score obtenu en allumant des fusées est multiplié par deux. Un combo se produit lorsque le joueur, après avoir réalisé une combinaison permettant d'allumer au moins une fusée, en réalise une deuxième d'affilée sans avoir à tourner une autre tuile de la grille. Mais étant donné la disposition des fusées et le sens de la gravité qui affecte les nouvelles tuiles, il est impossible de prévoir à coup sûr un combo, celui-ci dépend en grande partie de la chance, bien que le joueur puisse augmenter ses chances en disposant les tuiles comme il faut au préalable.

- Si les tuiles consumées transportent une médaille. Une médaille est une pièce qui se récupère en consumant la corde à laquelle elle est attachée. Une médaille bleue multiplie le score obtenu lors de l'allumage de fusées par deux, tandis qu'une médaille rouge multiplie le score par trois. Le coefficient multiplicateur des médailles se cumule, ainsi ramasser deux médailles rouges et une médaille bleue multiplie le score par dix-huit.

Cependant, ce jeu possède une petite spécificité, comparé aux autres prototypes que j'ai pu développer. Il possède une condition de défaite. Là où Frog Run permettait de se déplacer librement dans les divers niveaux, Flip and Snap et Shape Together étaient conçus de telle sorte qu'il est impossible de rester bloqué, Light the Fuse possède une mécanique qui peut entraîner la défaite du joueur. Chaque niveau offre à la disposition du joueur un certain nombre d'allumettes (oui encore des allumettes, mais en jeu la confusion ne se fait pas). A chaque fois que les fusées sont reliées aux allumettes en dessous de la grille par des cordes, une allumette est consommée, ce qui oblige le joueur à faire le plus de score possible à chaque coup pour remporter le niveau. Cependant, puisque les combos ne se contrôlent pas tout à fait, aucune allumette n'est consommée dans ce cas de figure.

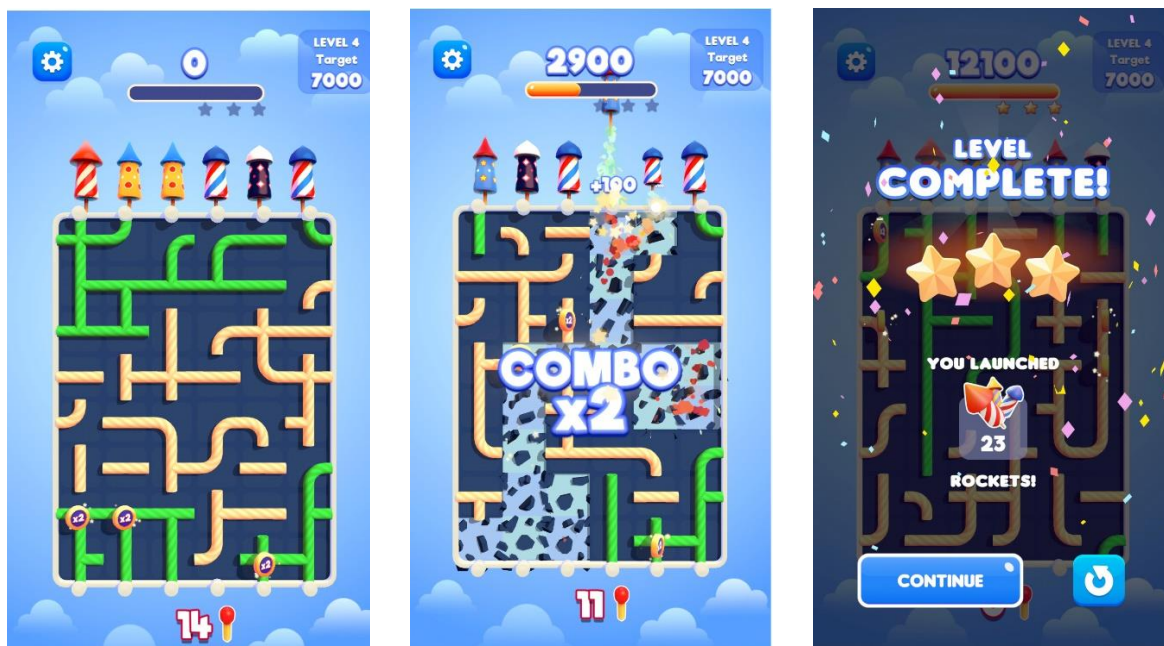


Fig 6 : Images tirées du jeu Light the Fuse

Le premier mois du développement de Light the Fuse, j'étais le seul à travailler sur ce projet. Il y avait plusieurs raisons à cela, la première est que nous avons trop de prototypes à développer simultanément, à la suite d'une vague de bonnes idées résultant d'un changement dans notre façon d'effectuer les brainstormings. La seconde est que l'entreprise a développé un tout nouveau dev-kit pour nous permettre d'accéder beaucoup plus facilement au code réutilisable sur nos projets, et puisque mes collègues étaient tous en train de développer un jeu pendant que je travaillais sur l'automatisation de la génération de build, j'étais le seul à ne pas encore avoir travaillé sur un projet avec le dev-kit. J'ai donc commencé seul Light the Fuse, le temps de m'accoutumer au dev-kit.

La première étape du développement, comme d'habitude, a été la création d'assets à base de formes primitives. La suivante a été la recherche de code réutilisable pour gagner du temps, et par chance, tout ce dont j'avais besoin pour créer les mécaniques de base était présent dans le nouveau dev-kit, que ce soit la gestion des inputs ou la génération et la gestion de la grille.

Lors de ma première semaine sur le projet, je me suis habitué au fonctionnement du dev-kit, en implémentant le système de grille, et les tuiles qui remplissent celle-ci. J'ai aussi ajouté la possibilité de tourner les tuiles lorsque le joueur touche l'écran, ainsi qu'une version très basique des fusées et des allumettes autour de la grille.

Le travail suivant, qui n'a pas été des moindres, a été l'implémentation de la logique du jeu, c'est-à-dire après chaque tuile tournée, exécuter un algorithme de pathfinding d'une part pour vérifier s'il existe un chemin reliant les allumettes aux fusées, et d'autre part pour colorer les cordes. Les cordes reliées de proche en proche aux allumettes sont colorées en jaune, et celles reliées de proche en proche aux fusées sont colorées en rouge (ces couleurs ont pu changer au fil des versions). Le but est que le joueur repère rapidement les parcelles de chemin déjà existant, afin d'éviter de les casser et voir quelles pièces tourner pour créer un chemin complet, ou alors au contraire le casser momentanément en attendant de pouvoir relier le plus de fusées possible d'un coup sans risquer de créer un chemin involontairement. Une fois ce pathfinding créé, il a fallu dans un premier temps détruire les tuiles ayant permis la création du chemin de part et d'autre de la grille, et ensuite faire apparaître de nouvelles tuiles au-dessus de la grille, et faire tomber le tout pour remplir la grille. En plus de cela, j'ai ajouté un masque, dont l'objectif est de cacher les tuiles tant qu'elles ne sont pas dans

la grille, pas soucis d'esthétisme. J'ai également désactivé les inputs dans la phase de vidage et de remplissage de la grille pour éviter l'apparition de bugs.

L'étape suivante a été un peu plus mathématique, j'ai dû réfléchir, selon la disposition des fusées et des allumettes s'il existe une configuration de tuiles telle qu'aucun chemin ne soit possible entre les allumettes et les fusées. Dans un premier temps, nous souhaitons mettre des fusées qu'au sommet de la grille et les allumettes en dessous, par soucis de clarté. Il s'est avéré que dans le cas où les allumettes couvrent toute une ligne et les fusées aussi, s'il y a plus de colonnes que de lignes dans la grille, il n'existe aucune configuration bloquante. En revanche, s'il manque au moins une fusée ou une allumette ou qu'il y a moins de colonnes que de lignes, un cas de figure bloquant est possible. Par exemple, si nous ne disposons que d'une seule allumette dans le coin inférieur gauche, et que toutes les tuiles au-dessus permettent de relier uniquement deux arêtes opposées de la tuile, à l'exception de la tuile dans l'angle supérieur gauche qui relie deux arêtes successives, et que sur la même ligne que cette dernière tuile, toutes les tuiles sont à nouveau des tuiles comme sur la première colonne. De manière générale, si toutes les entrées possibles sur une ligne ne peuvent pas accéder à la ligne au-dessus, nous sommes dans un cas de figure bloquant, et ceci n'intervient que si pour chaque allumette tous les chemins possibles mènent à une ligne où la première tuile atteinte est une corde reliant deux arêtes successives, et les autres tuiles de la ligne relient deux arêtes opposées. J'ai pour ce cas de figure précis rédigé un algorithme qui détecte les situations bloquantes et mélange les tuiles dans la grille pour débloquer la situation.

Une fois toute cette logique implémentée, j'ai été rejoint par un autre développeur et le designer sur le projet. L'objectif, maintenant que les mécaniques de base étaient fonctionnelles, était de construire les mécaniques plus profondes (le score et les médailles entre autres), et de rendre le jeu visuellement plus agréable, en ajoutant des plus beaux assets et des animations satisfaisantes.

Pendant que mon collègue développait une partie des animations, je me chargeais des quelques autres animations, ainsi que de quelques autres aspects visuels. Les deux points principaux de ce travail sont l'ajout de la possibilité de modifier le fond de la grille et d'y rajouter un contour directement depuis l'éditeur (le système de génération de grille se fait déjà depuis l'éditeur). L'autre point majeur a été l'ajout d'un nouveau type de pathfinding pour générer une flamme au bout de la ou les mèches qui se consume. Ce système est déjà implémenté, mais une amélioration est en cours au moment où je rédige ce rapport.

Voici donc l'ensemble de mon travail au cours de ces six mois de stage chez Blue Monkey Studio. Peut-être pourrai-je en dire plus sur ce dernier projet au cours de ma soutenance, puisqu'il me reste encore deux semaines de stage, largement de quoi finir cet ultime prototype.

VI. Bilan de la mission

1. Résultats

Premièrement, faisons un bilan sur les résultats du travail effectué en lui-même. J'ai aidé au développement de quatre prototypes de jeu au total, ainsi qu'à la première itération de l'un de ces prototypes. Deux de ces prototypes ont intégralement été terminés, il en va de même pour l'itération, et probablement un autre prototype sera terminé d'ici la fin du stage. A défaut d'avoir intégralement développé une solution pour le développement automatique des builds et leur publication sur l'App Store, j'ai réussi à implémenter une grande partie des fonctionnalités requises et ai beaucoup analysé les différents outils disponibles afin de sélectionner la meilleure option pour l'entreprise. Un concours a tout de même été remporté grâce à l'une des idées que j'ai pu soumettre à l'entreprise au cours de ce stage.

1.1. Analyse des résultats

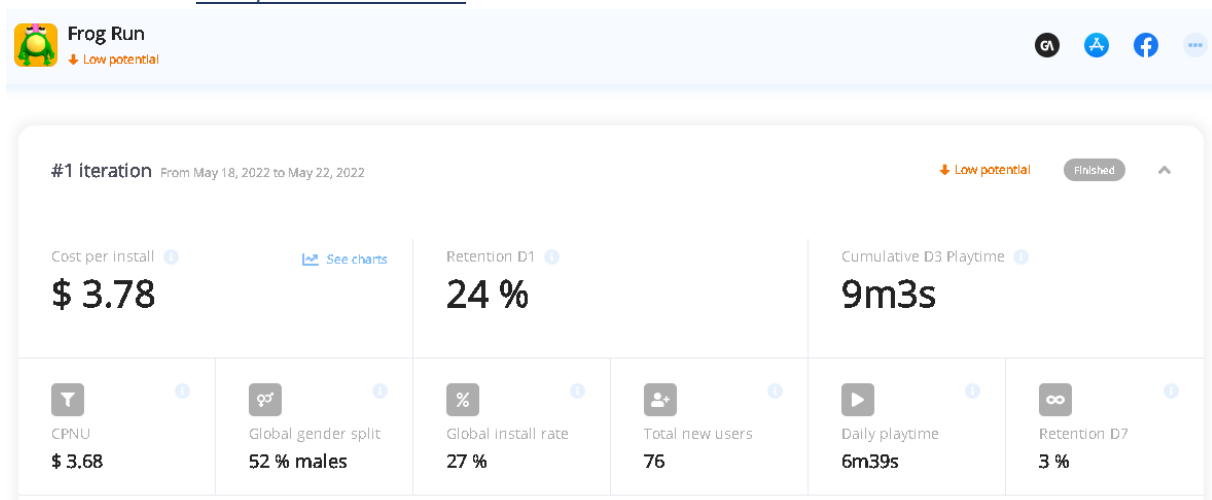


Fig 7 : Résultats obtenus par Frog Run

Avant d'effectuer l'analyse de ces résultats, je vais m'arrêter un peu sur les définitions de chacun des champs présents sur les rapports de Voodoo. Ces rapports sont mis à disposition des studios créateurs au bout d'une semaine de tests sur leur plateforme en ligne.

Le concept de cost per install (CPI) a déjà été décrit dans ce rapport. Lorsqu'il est trop élevé, c'est qu'il y a trop peu de joueurs sur le jeu, et les données récoltées ne sont alors pas cohérentes ou suffisantes pour être représentatives.

La rétention D1 est le pourcentage de joueurs qui ont lancé l'application le jour de l'installation. S'il est inférieur à 20 %, c'est un mauvais résultat, environ 30 % un bon résultat et au-dessus de 40 % un très bon résultat. Il permet de mesurer l'enthousiasme que les joueurs ont pour le jeu d'après les vidéos publicitaires et les captures d'écran.

Le cumulative D3 playtime correspond au temps de jeu moyen des utilisateurs sur les quatre premiers jours depuis l'installation. Il permet de mesurer la satisfaction du joueur. Plus il reste longtemps, plus le jeu est agréable à jouer et bien équilibré.

Le CPNU est le coût de la publicité divisé par le nombre de nouveaux joueurs. Plus il est élevé, moins le jeu est rentable.

Le Global Gender Split correspond à la répartition homme/femme des utilisateurs du jeu. C'est de manière générale une donnée dont on ne se sert jamais.

Le Global Install Rate est le nombre de fois que le jeu a été installé divisé par le nombre de clics réalisés sur les vidéos publicitaires. Plus le score est élevé, plus les vidéos CPI sont efficaces.

Comme son nom l'indique, le Total New Users correspond au nombre de nouveaux utilisateurs depuis la dernière version publiée du jeu. Elle n'est généralement pas utile en tant que tel, mais permet d'avoir un retour global sur le potentiel attractif du jeu.

Le Daily Playtime est le temps moyen qu'un utilisateur passe sur le jeu les jours où il lance l'application. Il permet de vérifier la vitesse de complétion des niveaux et l'engouement général du jeu lorsque comparé à la rétention D3. Un prototype réussi atteindra entre 10 et 30 minutes de jeu en fonction de sa taille et de son gameplay.

La rétention D7 est le pourcentage de joueurs qui lance le jeu sept jours après l'installation. Il permet de vérifier si les joueurs continuent à apprécier le jeu bien après l'avoir terminé, où en cas de faible rétention D1, si les joueurs n'ont vraiment aucun engouement pour le jeu et se mettent à y jouer vraiment tardivement.

Frog Run a obtenu des résultats très insatisfaisant, en particulier son CPI et sa rétention D1. Le jeu n'a pas su attirer l'attention du public malgré son originalité comparé aux autres jeux hypercasual. Les vidéos CPI ne sont pas à remettre en cause ici car celles qui ont réussi à attirer le plus de joueurs sont celles qui sont les plus fidèles au jeu. Probablement que le format mobile ne permet pas d'offrir au joueur suffisamment de liberté dans les mouvements, et donc pas assez de satisfaction, ce qui pourrait expliquer la faible rétention au cours des jours suivant l'installation.

Global ad videos (6) [Hide videos](#)







Performance	Spend	Impressions	Clicks	Installs	CPI
 #1 Performing video 9-ok.mov Download	\$225	56,193	230	55	\$4.09
 #2 Performing video 5-ok.mp4 Download	\$28	13,661	28	11	\$2.51
 #3 Performing video 3-ok.mp4 Download	\$16	4,389	14	6	\$2.66
 #4 Performing video 8-ok.MOV Download	\$7	1,422	4	2	\$3.56
 #5 Performing video 2-ok.mp4 Download	\$3	701	1	-	-
 #6 Performing video 1-ok.mp4 Download	\$2	526	1	-	-
Total	\$ 280	76 892	278	74	\$ 3.78

Fig 8 : Statistiques des vidéos CPI créées pour Frog Run

Sur ce tableau, on peut observer diverses statistiques sur les vidéos CPI créées pour le jeu. Ces statistiques permettent d'identifier quel aspect du jeu est le plus attrayant pour les joueurs, ainsi que la rentabilité du jeu comparé aux coûts publicitaires. Au cours de la semaine de test du jeu, l'éditeur verse une certaine somme d'argent dans la publicité du jeu. L'argent dépensé pour la diffusion de chaque vidéo correspond à la première colonne du tableau. Seulement, l'éditeur ne choisit que la somme totale à dépenser. Un algorithme se charge de la répartition en fonction du nombre de clics et d'installation, ce qui explique la grande différence d'impressions.

Ici, on peut observer qu'aucune vidéo de Frog Run ne permet de générer suffisamment de clics, et encore moins d'installation, pour assurer la rentabilité du jeu. Nous nous retrouvons donc face à un point mort. Mieux vaut abandonner le concept plutôt que de tenter une itération pour l'améliorer, ceci constituerait à coup sûr une perte de temps.

Il existe d'autres statistiques qu'il est possible de consulter sur la plateforme de Voodoo, mais celles-ci ont été bien moins importantes sur l'ensemble de nos prototypes pour comprendre ce qui a fonctionné ou non. Par exemple, on peut consulter le temps que les joueurs ont passé sur chaque niveau, à quel niveau ils ont généralement abandonné etc... Ces statistiques permettent d'analyser l'équilibre du jeu et son attrait.

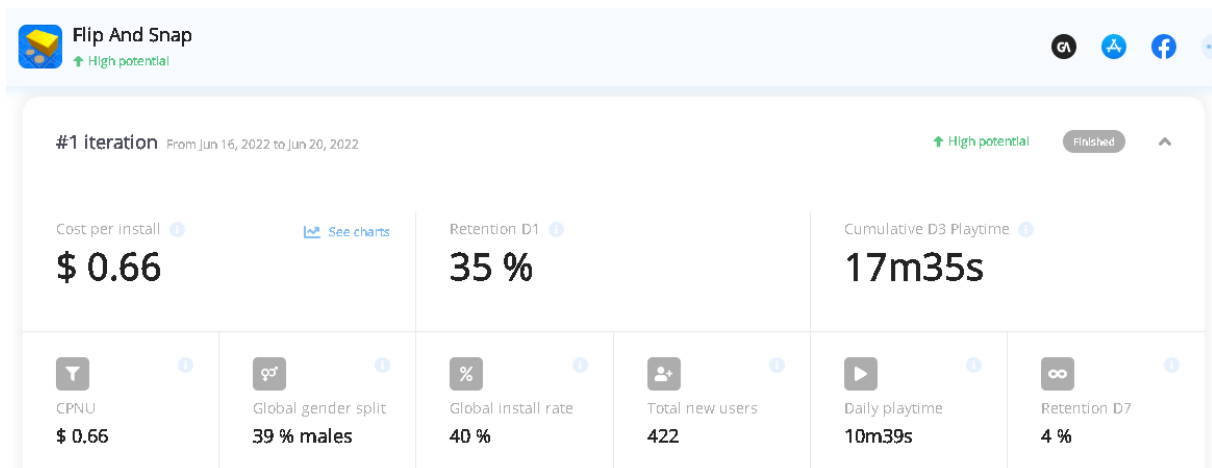


Fig 9 : Résultats de la première itération du jeu Flip and Snap

Contrairement à Frog Run, la première itération de Flip and Snap s'est avérée plutôt positive. Le jeu est pourtant un puzzle plus classique, mais avec un aspect nouveau grâce à l'ajout d'une troisième dimension au gameplay, qui a su attirer l'attention de nombreux joueurs. Ce qui explique le CPI particulièrement bon. De même la rétention D1 est bonne, les joueurs avaient envie de jouer au jeu tout de suite après l'avoir téléchargé et installé. Le temps de jeu cumulatif est au-delà d'un quart d'heure, et le Daily Playtime de dix minutes, ce qui reflète un jeu qui a su conserver l'attention des joueurs. C'est en observant ces statistiques sur le premier prototype que nous avons décidé d'essayer d'améliorer le jeu, en y ajoutant plus de trente nouveaux niveaux, une refonte graphique, un système de thèmes et de badge pour récompenser le joueur.

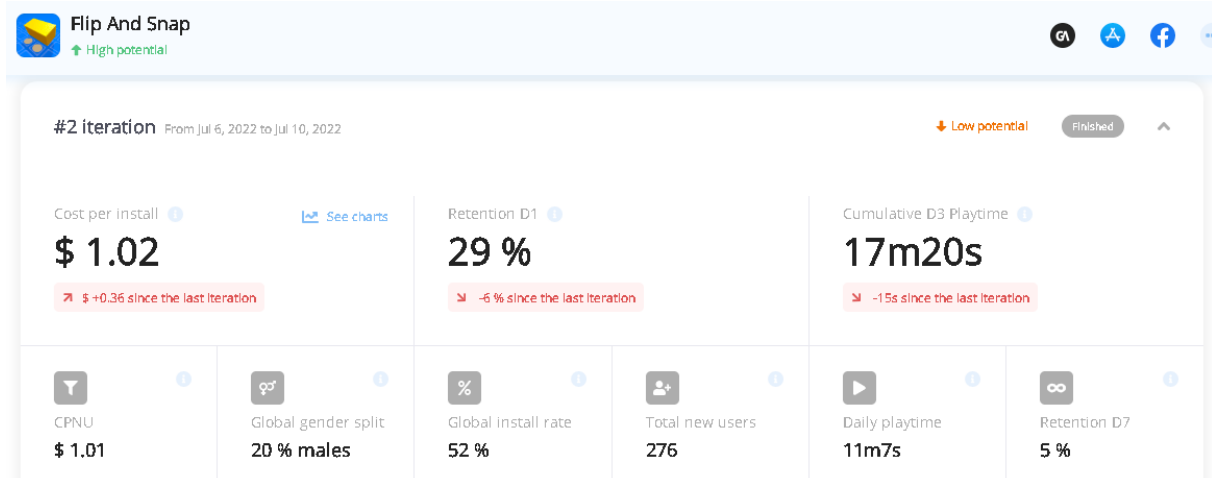


Fig 10 : Résultats de la deuxième itération du jeu Flip and Snap

La deuxième itération cependant, malgré l'amélioration de tous les aspects de la première version, a obtenu des résultats moins bons sur tous les aspects, à l'exception du Global install rate. Les statistiques ne sont pas non plus catastrophiques, surtout comparées à celles obtenues par Frog Run, mais sont tout de même tout juste trop mauvaise pour espérer le lancement d'un jeu rentable. Nous pensons que cette diminution au niveau du temps de jeu est dû au remaniement de la difficulté, là où la première itération était trop difficile, celle-ci est devenue bien trop facile, ce qui en plus de réduire drastiquement le temps passé sur chaque niveau crée de la lassitude chez le joueur. Nous avons donc décidé de créer une troisième itération sur laquelle je n'ai pas directement participé, où la difficulté a été ajustée, et plus de récompenses ont été distribuées au joueur, comme l'ajout de badges obtenus lorsqu'un thème est terminé. Pour ce qui est du CPI plus mauvais, alors que les vidéos sont les mêmes

que celles de la première itération mais de meilleure qualité visuelle, nous avons une hypothèse que nous cherchions à confirmer avec la troisième itération, et dont je reparlerai dans l'analyse de celle-ci juste en dessous.

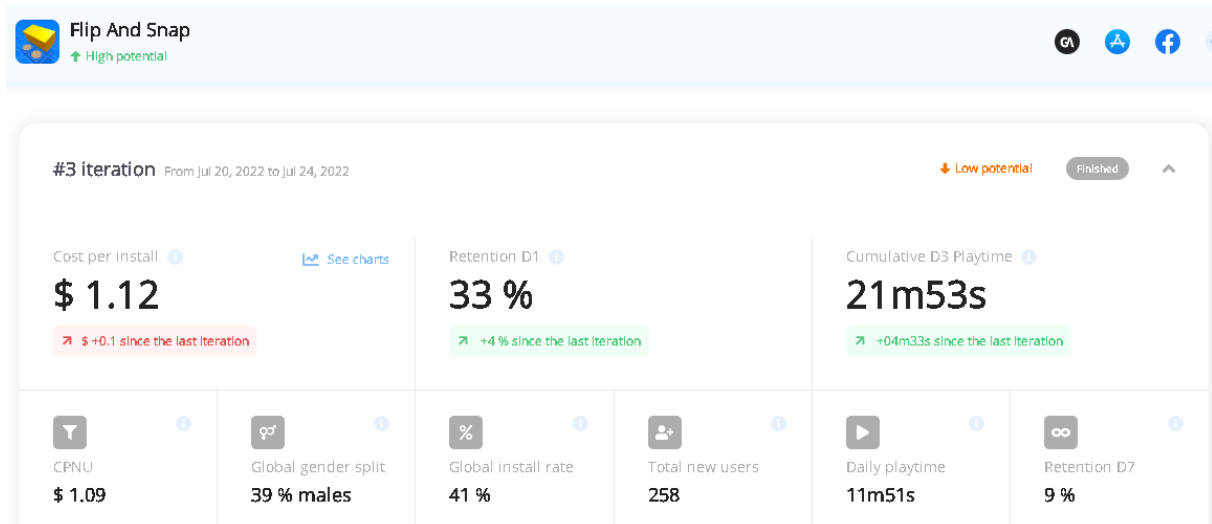


Fig 11 : Résultats de la troisième itération du jeu Flip and Snap

Lors de cette troisième itération, la rétention et le playtime sont dans l'ensemble meilleurs que jamais. Ce qui signifie que l'ajustement de la difficulté a effectivement été la cause des mauvaises statistiques en jeu de la version précédente. En revanche, le CPI est encore plus mauvais que lors de la deuxième itération. Ceci confirme l'hypothèse que nous avons. La première itération de Flip and Snap a été testée en mi-juin. Les deux autres versions, elles, sont sorties au cours du mois de juillet. Or le mois de juillet correspond au plein milieu des vacances scolaires américaines, lorsque le mois de juin n'en est que le tout début. Il est possible que, dans ce cadre de détente et de besoin de décompresser, la période estivale ne soit pas la meilleure pour jouer à des jeux de puzzle qui base leur intérêt sur la réflexion intense et la difficulté. Au contraire, les jeux d'actions et jeux de destruction (au gameplay dit brainless) ont bien plus la cote à ce moment de l'année. De manière générale, puisqu'il y a moins de personnes au travail pendant ces vacances, il y a moins de personnes dans les transports en commun notamment, et donc moins de joueurs potentiels. Nous espérons donc qu'en relançant une campagne de tests de Flip and Snap à une autre période de l'année, les résultats seront bien meilleurs.

Les autres jeux présentés au cours de ce rapport n'ont pas encore été testés ou les résultats des tests ne sont pas encore disponibles sur la plateforme de Voodoo au moment où j'écris ces lignes, peut-être aurai-je plus d'informations sur le sujet au cours de la soutenance.

1.2. Gestion des contraintes

Les diverses contraintes qui m’ont été imposées ont toutes été respectées.

J’ai toujours développé mes prototypes dans un temps raisonnable, à l’exception de Frog Run peut-être étant donné que c’était mon premier projet, qu’on a amplement sous-estimé la charge de travail nécessaire pour l’achever et que j’étais le seul et unique développeur tout au long du projet. Si nous avions été plusieurs développeurs dessus, j’aurais probablement produit un code plus propre, plus réutilisable pour l’avenir, nous aurions terminé le jeu plus tôt et probablement même que le prototype aurait été prêt pour être présenté au second concours organisé par Voodoo. Mais en ce qui concerne les autres projets, le temps de développement a été plus que raisonnable.

Une autre contrainte, ou plutôt une double contrainte, a été l’absence d’appareil compatible MacOS ou ios chez moi pour tester les builds ou tout simplement effectuer une build, ainsi que le manque de place sur mon ordinateur pour permettre le téléchargement et l’installation d’un bon émulateur. J’ai pu me débrouiller sans cependant, en créant pour mon compte des builds Android et non ios, et en les testant sur mon téléphone Android. Quand les tests sur un appareil ios étaient strictement nécessaires, j’ai communiqué l’information à mes collègues pour qu’ils puissent effectuer des tests pour moi.

1.3. Méthodologie et Choix techniques

Les technologies et choix techniques que l’entreprise m’a conseillé ou imposé se sont toujours avérés être de merveilleux alliés, mêlant gain de temps, accessibilité et correction facile en cas d’erreur. L’uniformisation des outils et versions de ces outils nous a aussi permis de travailler sans avoir constamment besoin d’adapter nos travaux. Les techniques nous ont aussi permis de travailler de façon plus efficace et de gagner beaucoup de temps pour l’avenir. Cependant, l’intégralité de ces méthodes et outils ont été imposés par l’entreprise, je ne peux donc pas faire d’analyse sur les outils que j’ai choisis puisqu’il n’y en a pas. Quoique, peut-être que deux outils font exception à la règle.

Le premier est Fastlane, que j’ai conseillé à mes collègues après avoir étudié les avantages et inconvénients des diverses solutions de publication automatique d’application sur les stores. Cependant, n’ayant pas pu tester moi-même l’efficacité de Fastlane, je ne peux pas réellement prendre du recul dessus.

En revanche, le second outil que j’ai eu la possibilité de choisir est Photofiltre7, que j’ai utilisé d’une part pour dessiner les pixelarts de Flip and Snap, et pour dessiner les schémas explicatifs des idées de jeux que j’ai présentées à l’entreprise. Étant donné que je connais ce logiciel depuis plus de 10 ans et que je l’utilise régulièrement, cela m’a permis de dessiner mes schémas et pixelarts rapidement et facilement sans avoir à apprendre à utiliser un autre logiciel potentiellement plus adapté. Paint aurait pu suffire pour les schémas, mais j’ai plus l’habitude de Photofiltre, il était donc préférable pour moi de l’utiliser par souci d’efficacité. En revanche pour le pixelart, Photofiltre n’est pas vraiment adapté, notamment parce qu’il ne permet pas de zoomer au-delà d’une certaine limite les images. Ce qui implique que mes pixelarts de dix pixels sur dix mesuraient au maximum cent soixante pixels de haut et de large, ce qui est bien trop peu et pas très ergonomique. Mais Photofiltre dispose d’un système de calque, qui m’a permis de dessiner des pièces superposées, puisque je dessinais à la fois la position initiale et la position finale des pièces. Le logiciel était donc complet pour le travail exigé. Mais il est

vrai que d'autres solutions complètes plus adaptées existent, comme Asesprite pour n'en citer qu'une. Mais le temps d'apprentissage de l'utilisation d'un autre logiciel n'aurait pas compensé la perte de temps due à l'utilisation d'un logiciel peu adapté. Un autre problème que j'ai pu avoir avec Photofiltre, mais qui est commun à tous les logiciels d'édition d'image utilisant des calques, est la lisibilité des dessins pour les personnes ne disposant pas du logiciel. Lorsque nous avons dû nous répartir les niveaux à designer, il était difficile pour mes collègues de comprendre comme certains niveaux étaient agencés.

2. Retour d'expérience

2.1. Recul sur les Résultats mission

Dans l'ensemble, je suis très satisfait des résultats qu'on a pu obtenir sur les jeux que j'ai au minimum aidé à développer. Bien qu'aucun prototype (du moins pour l'instant) n'ait su devenir un hit, ce sont tous des jeux qui font très professionnel (parce qu'ils le sont j'imagine), et bien que n'ayant pas percé, avaient et ont toujours malgré tout un bon potentiel, même s'ils n'ont pas trouvé leur public dans leur version prototype. J'aurai beaucoup aimé aller plus loin sur le projet de déploiement automatique, mais les contraintes techniques ont fait que je n'ai pas pu aller jusqu'au bout de la chose. J'ai malgré tout réussi, notamment sur Unity Clouds, à développer une grande partie des fonctionnalités désirées sur cette mission. Enfin, même si rien n'a abouti à une version définitive, une bonne partie du code que j'ai écrit sera ou est déjà transféré dans le dev-kit, et sera donc toujours utile à l'entreprise, on ne peut en aucun cas dire que le travail que j'ai effectué n'aura servi à rien, et je suis heureux d'avoir pu faire tout ce que j'ai fait.

En développant les différents scripts des jeux, j'ai appris à perfectionner mon sens de l'organisation et l'élaboration de bonnes structures de code suivant des principes essentiels pour garantir l'efficacité de celui-ci.

En cherchant des idées de jeu, j'ai travaillé sur mon esprit créatif et la recherche de solutions innovantes.

En cherchant à gagner le plus de temps de développement possible, j'ai appris à économiser le plus possible les ressources à disposition et à adapter les fonctionnalités de mon travail en fonction des contraintes de temps.

En utilisant les différents outils mis à disposition par l'entreprise, j'ai travaillé ma communication avec les autres membres de mon entreprise afin de savoir quel est exactement le rôle de chacun dans l'élaboration des projets, ainsi que les rôles des personnes extérieures à l'entreprise qui agissent directement sur nos prototypes ou sur l'organisation de l'entreprise. J'ai aussi appris l'utilisation concrète de nombreux outils de gestion de projet.

2.2. Difficultés

Vous l'aurez compris en lisant ce rapport, sur de multiples aspects, ma principale difficulté a été de ne pas posséder d'appareil de type Mac Os ni ios, notamment pour pouvoir travailler

convenablement sur le projet de déploiement automatique, au ne serait-ce que pour pouvoir tester les builds des projets parallèles aux miens et de faire mes retours dessus.

Parmi les difficultés suffisamment importantes pour être relevées, je dirai qu'il y a eu le tout premier prototype sur lequel j'ai travaillé. Au début, l'organisation du travail n'était pas la même au sein de l'entreprise. J'ai travaillé seul sur un projet particulièrement retors, sans pouvoir réutiliser le code déjà écrit de précédents projets, certes en ayant un bon accompagnement et de bons conseils au cours du développement. Mais Frog Run était décidément un projet trop ambitieux pour un premier jeu, qui a particulièrement contrasté avec le projet Flip and Snap, qui au contraire était plutôt simple à développer. Malheureusement, puisque nous souhaitions à l'origine faire participer le prototype de Frog run au second concours organisé par Voodoo, nous n'avions pas voulu décaler son développement pour commencer le développement de prototype de jeu hypercasual moins exigeant et ambitieux.

Par la suite, je n'ai pas eu réellement de difficulté majeure, seulement le genre de difficulté classique avec un fichier de code assez complexe qui génère des bugs sans qu'on arrive à comprendre pourquoi ni comment (la génération de nouvelles tuiles pour la grille de Light the Fuse, c'est de toi que je parle).

Une autre difficulté que je peux citer peut-être est l'ensemble des limitations en termes de mémoire de mon ordinateur. Avec les projets Unity en plus, je n'ai presque plus de place disponible sur les deux disques durs de mon ordinateur, en tout cas pas de quoi installer un bon émulateur Mac Os ou Ios pour faire les tests dessus lorsque nécessaire. De plus, ma mémoire RAM est elle aussi faiblarde, il n'est pas rare qu'Unity a planté au cours des tests que j'effectuais sur les prototypes à la recherche d'éventuels bugs. Parfois même, j'ai assisté à certains bugs particulièrement étranges, comme une fonction qui ne s'exécute pas alors qu'elle le devrait, qui ne se sont jamais produits chez mes collègues, ni sur leurs appareils mobiles une fois la build du jeu construite. J'ai donc attribué ces bugs à ma mémoire RAM tremblotante.

2.3. Apports personnels

Les apports de ce stage sur mon expérience professionnelle sont multiples.

J'ai réussi à obtenir ce stage après avoir posté ma candidature, et avoir développé un prototype de jeu qui a fait office d'entretien technique. Si j'ai pu réussir à créer suffisamment rapidement un prototype de jeu pour obtenir ce stage, c'est parce que je possédais déjà au préalable des connaissances que j'estime solides dans le domaine du jeu vidéo et de toutes les étapes de sa création. J'ai pendant cinq ans développé des prototypes avec mes amis durant mes week-ends, étudié comment les gros studios du jeu vidéo appréhendent certaines questions très techniques, comme la simulation de la mer, de l'eau en général, les algorithmes de pathfinding, le foliage (l'ajout de végétation dynamique ou non de façon plus ou moins procédurale)... J'ai beaucoup joué à des jeux développés par des toutes petites équipes et cherché comment malgré le manque de moyens, de personnel et de temps, ces jeux arrivaient à captiver le joueur aussi bien que de grosses productions, avec souvent une dose folle d'inventivité et d'originalité en plus. J'ai même beaucoup étudié les jeux flash, qui représentent la partie extrême des jeux sans budget, souvent hideux, mais qui ne conservent que l'aspect fun d'un jeu vidéo. Cependant, je ne connaissais pas du l'hypercasual. Du moins, je savais de quel genre de jeu il retournait, j'en ai moi-même testé quelques-uns bien avant le début du stage. Mais j'ignorais que ce type de jeu demandait un développement si contraint et était soumis à autant

de règles, contrairement aux jeux que j'ai plus l'habitude de tester qui sont par essence sans aucune autre limite que les avancées technologiques de leur époque. Découvrir qu'il existait un autre monde (j'ose le dire) que celui que je connaissais bien dans le jeu vidéo à grandement attisé ma curiosité et affiné mon esprit critique. J'ai pu me rendre compte au cours de ce stage que j'ai assimilé pleins de codes et concepts propres à l'hypercasual, qui pourraient plus tard me servir dans d'autres domaines du jeu vidéo. Citons par exemple l'intuitivité extrême de tous les aspects du jeu hypercasual. Dans un jeu classique, un objet de soin sera représenté par une potion ou un cœur, car ce sont des concepts amenés par les premiers jeux vidéo qui ont perduré dans l'histoire et sont devenus des conventions généralisées à tous les jeux ou presque. Sauf que ces potions et cœurs sont très majoritairement rouges, et le rouge représente généralement quelque chose de négatif. C'est pourquoi dans le domaine de l'hypercasual les objets de soin ne sont jamais représentés ainsi. On va plutôt utiliser des croix vertes rappelant le symbole de la pharmacie pour évoquer des objets de soin. Dans les jeux classiques, des confusions se font parfois à cause justement de ces conventions, là où par principe cela n'arrive jamais dans les jeux de l'hypercasual.

Un deuxième apport de ce stage a été la meilleure compréhension du rapport entre l'éditeur et le studio de développement. Jusqu'ici je n'avais qu'une vague idée de ce qu'était exactement un éditeur, et disposais de mes propres préjugés sur le sujet. La relation entre Blue Monkey Studio et Voodoo m'a aidé à comprendre concrètement l'impact d'avoir un éditeur pour un studio et l'impact sur le développement des jeux. Par exemple, à quel point l'éditeur intervient dans la vie de l'entreprise. Chez Blue Monkey Studio, les fondateurs se réunissaient avec un représentant de Voodoo une fois par semaine pour parler des idées de jeu sélectionnées par les membres du studio afin d'avoir le point de vue d'une personne connaissant mieux que nous le marché, et qui aura un point de vue bien plus critique que nous sur chaque concept, qui verra bien plus facilement des axes d'amélioration, et nous proposera son feu vert pour le développement du jeu s'il sent un gros potentiel. L'entreprise présente aussi les builds de ses prototypes en cours pour vérifier si nous partons dans la bonne direction, qu'il n'y a pas eu aucun problème de communication lors de la présentation des concepts et des échanges de points de vue. Parfois, le représentant de Voodoo pourra aussi prodiguer des conseils sur d'autres aspects du jeu à implémenter. C'est par exemple un représentant de chez Voodoo qui nous a suggéré de remplacer la grenouille par un bonhomme pour réaliser certaines vidéos CPI pour Frog Run.

Au cours de ce stage j'ai été amené à manipuler des technologies que je connaissais plutôt bien, que je pensais même maîtriser parfois. C'est notamment le cas du moteur de jeu Unity. Or, l'entreprise m'a fait découvrir des façons de coder un jeu différemment et bien plus efficacement que ce que j'avais l'habitude de produire, au travers de principes de code et de design patterns qui m'étaient inconnus. J'ai ainsi appris à maîtriser des techniques de développement nouvelles qui sont en tout point meilleures que celles que j'utilisais auparavant.

Mais j'ai aussi découvert des technologies que je ne connaissais pas du tout, et qui constituent un gain de temps considérable. C'est notamment le cas de Sourcetree. Lorsqu'avant j'utilisais Git Bash et Git Hub pour gérer mes répertoires et branches sur Git, l'entreprise m'a vivement suggéré l'utilisation de cet outil qui certes n'est pas parfait, loin de là, mais facilite grandement la gestion de conflits et dispose de toutes les fonctionnalités les plus essentielles disponibles sur Git Bash, tout en étant bien plus intuitif et simple d'utilisation. J'ai également découvert l'éditeur de code JetBrains Rider, qui est plus adapté que Visual Studio Code pour travailler sur des projets Unity, mais qui dans mon cas ne changeait rien puisque de mon côté l'application était mal configurée. Mais avec une bonne configuration, j'ai pu observer que cet éditeur pouvait grandement simplifier la vie du développeur.

2.4. Aspect sociaux-environnementaux de ce stage

Au courant de ce stage, j'ai été soumis à diverses problématiques. Des problématiques liées aux décisions de l'entreprises pour répondre à des questions qui nous concernent tous.

En termes d'éthique, l'entreprise a mis un point d'honneur pour écarter totalement les thèmes choquants ou stigmatisants pour accompagner le gameplay de ses jeux. Notons que cette décision provient bien de l'entreprise elle-même et non d'une réticence de la part de l'éditeur, puisque malheureusement, les thèmes oppressifs peuvent s'avérer très vendeurs. Par exemple, un jeu a été publié il y a deux ans de cela, et consistait à faire courir un personnage qui grossissait en mangeant des hamburgers jusqu'à ne plus courir mais rouler. Blue Monkey Studio refuse d'évoquer des thèmes traitant de manière stigmatisante l'obésité, le racisme ou le sexisme par exemple pour faire vendre ses jeux. Tous les jeux proposés sont moralement neutres, et ne prennent pas partie de quelque opinion que ce soit.

L'entreprise étant très petite et récente, et ne disposant à son actif qu'un seul jeu extrêmement populaire, elle possède une influence très limitée. Son image d'un point de vue extérieur est peu resplendissante car édulcorée par l'éditeur. Dans le domaine des jeux mobiles, les jeux sont toujours associés aux éditeurs et non aux studios de développement sur les plateformes de commercialisation ou de diffusion. Par exemple, le nom Voodoo apparaît sur les stores et non celui des studios de développement. Si on veut savoir aisément quel studio a développé le jeu, il faut télécharger le jeu et le lancer. On peut ainsi voir le logo de l'entreprise à l'ouverture de l'application, et s'il y a des crédits disponibles, connaître l'équipe de développement. Ainsi, puisque son nom est rarement mentionné publiquement, le studio ne souhaite pas porter une attention particulière à son image sur d'autres aspects que le jeu vidéo.

En termes de sécurité, tous les mots de passes sont générés via l'application KeePass2. C'est un gestionnaire de mot de passe qui, en lui renseignant un mot de passe particulièrement complexe mais unique, va permettre à l'utilisateur d'accéder à une base de données contenant des mots de passes générés de façon à être inviolables en un temps raisonnable, associés chacun à une plateforme, un compte, une application etc... Les droits d'accès sur les projets et autres contenus de l'entreprise sont sécurisés via les plateformes concernées (Github etc...) et ne sont en aucun cas communiqués à des personnes extérieurs, sauf dans certains cas, comme pour la consultation de politique de confidentialité des jeux ou pour des données partagées avec l'éditeur.

En termes de santé, l'entreprise a décidé de n'opérer qu'en distanciel uniquement. Ainsi, les chances de propager des maladies comme le COVID19 sont largement amoindries.

Enfin, en termes d'environnement, l'entreprise est encore trop récente et trop petite actuellement pour que cela constitue une problématique majeure. Les seuls points positifs pour l'environnement sont des phénomènes secondaires découlant des décisions de l'entreprise. Par exemple, le télétravail permet aux salariés de travailler sans avoir à emprunter un moyen de transport, et donc de moins polluer, bien que la problématique principale résolue par cette décision soit une problématique de santé. De même, lorsque l'entreprise souhaite développer des jeux en multijoueur, elle développe en réalité une illusion de multijoueur par l'intermédiaire de comportements réalistes des adversaires et de l'utilisation de faux pseudonymes. Cela permet non seulement une économie de temps, mais aussi une économie de moyen puisqu'il n'est alors pas nécessaire de posséder de serveurs pour héberger les parties multijoueur. Et donc, par effet domino, l'entreprise pollue moins puisqu'elle ne consomme pas l'énergie nécessaire pour faire tourner des serveurs en permanence.

2.5. Futur immédiat

Dans l'idéal, je cherche à obtenir un emploi dans le domaine du jeu vidéo au niveau ingénieur. Mon domaine de compétences étant très large, je serai d'autant plus satisfait si mon futur emploi me permettra de travailler sur diverses phases de la conception d'un jeu vidéo, comme ça a été le cas lors de ce stage, qui m'a plus que conforté dans mon choix de travailler dans ce domaine. Mais un emploi consistant à ne travaillé que sur une phase ou un aspect du projet de jeu dans sa globalité me conviendra très bien aussi (par exemple, en ne travaillant que sur la conception d'un moteur de jeu ou de divers éditeurs, ou encore en travaillant sur l'aspect graphique, les shaders...).

Si hélas je ne parvenais pas à m'intégrer professionnellement dans le domaine du jeu vidéo, je postulerais en priorité pour des emplois similaires, par exemple dans le cinéma d'animation qui utilise de plus en plus Unity. Et si encore une fois je ne trouve rien, je chercherai un emploi dans le développement en général, sans plus de restrictions.

2.6. Perspective sur deux ou trois ans

J'aimerais, en parallèle de mon travail, développer un jeu (complet et commercialisable cette fois) avec mes amis et potentiellement d'autres personnes passionnées, en exploitant tout ce que j'ai appris jusqu'ici. Mon idéal serait d'être capable, peut-être avec l'aide d'un éditeur, de commercialiser ce projet entre amis de sorte qu'on puisse former notre propre entreprise. Cet idéal nous demandera beaucoup de travail sérieux et très réfléchi. Pas question par exemple de développer un jeu parce que ce concept nous plaît, mais bien parce que le concept plaira au plus grand nombre de joueurs adeptes (notre public cible raisonnable).

Si le succès n'est alors pas au rendez-vous, je continuerai (ou débiterai si ça ne s'est pas déjà fait avant) ma carrière dans le jeu vidéo, que ce soit dans une grande ou moyenne entreprise. Ma préférence étant toujours portée vers les moyennes et petites entreprises, car elles laissent bien plus la parole à leurs employés qui ont des idées plein la tête.

2.7. Mon cycle ingénieur

Au cours de ces trois dernières années de formation à Polytech, j'ai appris à prendre le temps de planifier mes projets, d'analyser les solutions déjà existantes et de m'en inspirer pour les réadapter et gagner du temps. J'ai développé mes aptitudes dans la programmation, me permettant de coder de façon bien plus ergonomique et optimisée. J'ai compris l'importance capitale de la communication au sein d'une équipe tout au long du projet. Et part dessus tout, je pense que Polytech m'a permis d'acquérir d'une part des connaissances suffisantes dans la majorité des domaines de l'informatique pour ne pas être perdu lorsque je travaille en collaboration avec des équipes venant de différents domaines, et d'autre part de développer mon esprit critique et d'analyse sur les résultats, afin d'éviter de commettre en boucle les mêmes erreurs à l'avenir.

VII. Conclusion

Au cours de ces six derniers mois, j'ai pu travailler dans mon domaine de prédilection, le jeu vidéo. J'ai découvert de nouvelles technologies et méthodes de travail, ainsi qu'un marché spécifique du jeu vidéo dont je ne connaissais pas la teneur. Ce stage a été pour moi probablement le plus utile de tous ceux que j'ai pu réaliser, et m'a conforté dans mon choix de travailler dans ce domaine. J'ai pu prendre conscience du rôle que peut avoir un ingénieur sur un projet, même miniature. Je suis donc au moment où j'écris ces lignes à la recherche d'un emploi de niveau ingénieur dans le jeu vidéo, quel que soit le domaine concerné, et aussi spécifique soit-il (travailler sur un moteur de jeu, sur les algorithmes de calcul de la lumière, sur des intelligences artificielles et j'en passe...). J'espère pouvoir utiliser toutes les connaissances acquises au cours de ce stage et de ces cinq dernières années chez Polytech pour convaincre mon futur employeur de m'embaucher.

VIII. Lexique et sources

1. Vocabulaire de l'hypercasual

ASMR : pour Autonomous Sensory Meridian Response, l'ASMR désigne à l'origine la sensation agréable de picotement que certaines personnes peuvent ressentir à l'intérieur des oreilles jusqu'au milieu du dos en entendant certains sons, comme un chuchotement ou un tapotement sur un clavier d'ordinateur. Par extension, une sensation ASMR désigne toute sensation très agréable et satisfaisante, qu'il s'agisse d'une sensation auditive, visuelle ou sensitive dans le cas d'un jeu vidéo mobile.

Bundles : les bundles sont des ensembles de ressources (des scripts, des modèles...) qui ne sont pas propre à un projet spécifique. On peut créer un bundle si on veut partager des ressources sur l'Unity Asset Store par exemple. Le contenu du bundle doit, du moins en théorie, être indépendant, c'est-à-dire facile à implémenter dans tout autre projet sans avoir à adapter son code ou sa hiérarchie de gameObjects. Chez Blue Monkey Studio, nous utilisons par exemple divers bundles regroupant notre code réutilisable.

Classe Manager : une classe manager est une classe qui a pour objectif de gérer un aspect très spécifique du jeu. Par exemple, un level Manager va gérer l'enchaînement des niveaux du jeu, et un UI Manager va s'occuper de toutes les interfaces utilisateur, qui ne servent pas au déroulement du jeu mais à l'information au joueur de certains paramètres.

Easing : L'easing correspond à la variation de vitesse d'un mouvement ou d'une animation. Cette variation est paramétrée via une courbe temporelle, et a pour but de fluidifier le mouvement d'un

objet. L'easing est notamment utilisé lors des translations et rotations afin d'obtenir un visuel plus nerveux ou entraînant.

GameObject : Un gameObject est la classe de base de tout objet sur Unity. Dans une scène, tout est composé de gameObjects. On peut y intégrer des scripts, un modèle, un matériau, des boîtes de collisions, de l'émission lumineuse, des particules, et toute autre sorte de composants.

Gameplay : l'ensemble des caractéristiques propres à un jeu vidéo. Le gameplay est la façon de jouer à ce jeu. C'est l'ensemble composé de son objectif, ses mécaniques, et son intrigue s'il y en a une. Le gameplay est opposé à la direction artistique, qui fait référence à l'aspect artistique et sensoriel du jeu, les graphismes, les effets sonores etc... L'imagination, l'étude et la conception du gameplay est appelée le game design.

Hit : un hit est un grand succès. Dans le domaine de l'hypercasual, on parle de hit quand un jeu est téléchargé plusieurs millions de fois sur les stores sur lesquels il est disponible.

Hypercasual : L'hypercasual est un sous domaine du jeu vidéo. Il fait partie des jeux casual, qui sont caractérisés par un public cible sans limites et un gameplay relativement simple mais efficace. L'immense majorité des jeux mobiles font partie du domaine des jeux casual, et au contraire très peu de jeux casual ne sont pas sur mobile, on peut donc raisonnablement rajouter le support du jeu comme étant une caractéristique du domaine. L'hypercasual pousse les caractéristiques du casual à l'extrême, en ajoutant ses propres contraintes, comme la rapidité de développement, la restriction de la complexité du jeu, de ses visuels et de ses mécaniques.

Input : Les inputs sont les diverses interactions que le joueur peut avoir avec l'appareil sur lequel le jeu est installé, afin d'avoir un impact sur le déroulement du jeu. Sur un appareil mobile, ces inputs sont le touché, le glissé ou le zoom sur l'écran, les divers boutons ou encore le microphone. Les divers inputs sont généralement divisés en plusieurs sous catégories. Par exemple, en glissant le doigt sur l'écran, on peut effectuer un Drag and Drop pour déplacer un objet par exemple, mais aussi un Draw pour dessiner une forme.

Level design : le level design correspond à la conception des niveaux qui composent le jeu, en exploitant au mieux les diverses mécaniques que le jeu propose. Par exemple si le jeu propose un déplacement du personnage via des sauts, le level design peut être la construction d'un niveau très vertical. Si le jeu propose de faire exploser des objets, il faut veiller à bien les écarter les uns des autres par soucis de visibilité. Toutes ces questions sur l'édition d'un niveau fait partie du level design.

Mesh : Un mesh est l'ensemble des faces qui constituent un modèle. Sur ces faces, on peut appliquer des textures (la couleur sur les modèles), des normal maps (qui permettent de simuler un relief à petite échelle), et pleins d'autres images exerçant une influence sur le rendu visuel. Il est possible via script de changer la position des sommets constituant un mesh, ce qui permet par exemple de simuler des vaguelettes sur un plane (un simple carré prédécoupé en plusieurs centaines de faces).

MonoBehaviour : Un MonoBehaviour est un type de script dont le principe est de ne gérer qu'un unique comportement. Par exemple, si on souhaite qu'un objet tourne sur lui-même et attire à lui d'autres objets tel un aimant, alors il faudra d'une part lui ajouter le MonoBehaviour « Rotate », et d'autre part le MonoBehaviour « attract ».

Pathfinding : Le pathfinding est la recherche automatique d'un chemin possible ou optimal selon la situation. Typiquement, on peut utiliser un pathfinding sur un labyrinthe ou tout autre puzzle pour vérifier si ce labyrinthe peut être résolu, voire, si c'est le cas, pour trouver le chemin optimal.

Pixelart : Le pixelart est une technique de dessin qui consiste à n'utiliser qu'une résolution d'image très réduite. Par exemple dans un jeu vidéo, un personnage dessiné en pixelart ne mesurera qu'entre vingt et cent-vingt pixels de hauteur. Il n'y a pas une limitation de résolution précise pour le pixelart, les caractéristiques typiques de cette technique sont la facilité à pouvoir distinguer clairement ce qui est représenté malgré la faible résolution, ainsi que la présence de pixels apparents. Dans le jeu vidéo, le pixelart est souvent utilisé pour donner un côté rétro aux graphismes.

Prefab : un prefab est un gameObject enregistré dans les fichiers du projet Unity, avec tous ses composants et les valeurs exactes de ceux-ci. Il peut être réutilisé dans une scène pour générer plusieurs instances d'un même gameObject, par exemple, si on a besoin dans un puzzle de générer plusieurs pièces, alors on enregistre un prefab de pièce, et on le duplique dans la scène. Lorsqu'on modifie une valeur quelconque ou qu'on ajoute un composant au prefab, toutes les instances de ce prefab sont automatiquement mises à jour. Il est possible d'établir une hiérarchie entre les prefabs, afin qu'un prefab dépende d'un autre et soit mis à jour lorsque son parent est changé. Dans notre exemple de puzzle, cela permet de créer plusieurs pièces de puzzle différentes, qui restent toutes des pièces avant tout, mais chacune avec une spécificité.

Principes SOLID : Les principes SOLID sont cinq grands principes adaptés à la programmation orientée objet (l'approche de la programmation adoptée par le moteur Unity). SOLID est un acronyme pour :

-Single responsibility : une classe ou une méthode ne doit avoir qu'une seule responsabilité, ne doit implémenter qu'un seul comportement (c'est le principe de base du MonoBehaviour).

-Open/closed : les classes et méthodes doivent être fermées à la modification, mais ouvertes à l'extension. C'est la base du code réutilisable. On utilise pour cela au mieux l'héritage et les interfaces lorsque nous développons des classes.

-Liskov substitution : soit un type T1, et T2 un sous-type de T1, alors T2 doit pouvoir remplacer T1 partout sans que cela ne perturbe la cohérence du programme. On applique ce principe aussi bien au code développé qu'à la hiérarchie de **prefabs**.

-Interface segregation : mieux vaut disposer de plusieurs interfaces spécifiques que d'une unique générale. C'est un principe que nous appliquons notamment sur les GameLogic, du code qui est très spécifique au jeu auquel il y est associé.

-Dependency inversion : il faut dépendre des abstractions et non des implémentations. Il faut donc le plus possible utiliser des interfaces plutôt que des classes dont le comportement peut varier.

Ragdoll : la ragdoll est le comportement physique ajouté aux membres et autres parties du corps d'un personnage (humanoïde ou non), donnant l'illusion que ses muscles ne fonctionnent plus. La ragdoll fonctionne grâce à un système de squelette implémenté dans le modèle du personnage. Chaque articulation de ce squelette (étrangement appelée os) est paramétrée pour laisser certains degrés de liberté aux faces du modèles qui en dépendent. Cela permet ainsi de simuler le comportement d'un cadavre ou d'une créature inconsciente soumis à diverses forces, comme la gravité ou une explosion.

Rétention : la rétention correspond au temps durant lequel un joueur va jouer au jeu. Il existe plusieurs mesures de la rétention possible, qui sont détaillées dans la partie analyse des résultats de ce rapport.

Runner : un runner est un type de jeu très spécifique et très populaire dans les domaines du jeu casual et hypercasual. Dans un runner, le joueur contrôle un personnage en course sur une piste linéaire très majoritairement générée aléatoirement. Cette piste est semée d'obstacles en tous genre à éviter en utilisant la mécanique principale que propose le jeu, par exemple en ramassant suffisamment d'objets en chemin, ou simplement en glissant le doigt sur l'écran. Le but d'un runner est d'aller au bout du parcours s'il y en a un, ou au moins d'aller le plus loin possible. La caméra sur ce genre de jeu est placée derrière le personnage pour permettre de voir la piste devant lui et de le faire se déplacer sur les côtés, ou à l'arrière du personnage légèrement sur le côté si la profondeur et la verticalité sont plus importantes.

Scène : une scène dans Unity est un ensemble de gameObject dans une disposition particulière. Dans une scène on retrouvera souvent le personnage que le joueur contrôle, les décors, les objets avec lesquels on peut interagir, et les managers propres à la scène. Par choix, on peut assimiler un niveau à une ou plusieurs scènes, ou ne disposer que d'une seule scène avec un manager qui va charger des Prefabs correspondant aux niveaux.

Store : Un store est une application faisant l'inventaire des jeux et applications disponibles à l'installation sur l'appareil mobile concerné. Ils permettent le téléchargement et potentiellement l'achat d'autres applications. Sur Android, le store principal est le Play Store de Google, tandis que sur ios le store le plus populaire est l'App Store.

Thème : Le thème d'un jeu correspond à l'univers dans lequel le joueur sera plongé. Il doit être évocateur et cohérent pour fonctionner. Par exemple, si le jeu se base sur un thème pirate, on peut y retrouver des mers, des navires et des trésors, à la condition tout de même que ces décors ou objets servent le gameplay. Le thème accorde de la cohérence au jeu et offre des repères au joueur.

Vidéo CPI : une vidéo CPI (pour Cost Per Install) est une courte vidéo dont l'objectif est d'estimer le potentiel d'un jeu, en en faisant la publicité sur internet. Une vidéo CPI doit être attractive et capter l'attention de l'utilisateur dès les premières secondes. Elles sont diffusées sur les réseaux sociaux ou sur les emplacements publicitaires d'autres jeux vidéo mobiles. Lorsqu'on crée une vidéo CPI pour un

jeu hypercasual, on en crée en réalité plusieurs, cinq au grand minimum, montrant toute un aspect différent du jeu, que cet aspect soit réellement implémenté dans le jeu ou non. En analysant le nombre de clics qu'une vidéo CPI a généré, on peut la comparer aux autres vidéos CPI publiées afin d'observer les tendances actuelles et les aspects les plus attrayants du jeu, afin d'en faire la meilleure publicité possible à l'avenir. Cela permet aussi à l'éditeur d'estimer le coût par installation du jeu s'il venait à être publié. Plus une vidéo CPI est attrayante, plus le public visé va cliquer sur les publicités, et plus faible sera le coût par installation. En général, les éditeurs cherchent des jeux dont le coût par installation est largement inférieur à un dollar.

2. Logiciels et plateformes utilisées au cours du stage

Apple Store Connect : L'Apple Store Connect est une interface permettant aux développeurs de publier et monitorer des applications sur l'App Store. Ils peuvent ainsi publier des mises à jour de leurs applications, observer l'évolution du nombre de téléchargements, et autres données relatives à leurs applications.

Dotween : Dotween est un outil gratuit et open source permettant d'effectuer animations orientées objet sur Unity. Il permet de gérer les différentes variations des valeurs propres à une animation, notamment sa vitesse, ainsi que le séquençage entre plusieurs animations. Sa fonctionnalité la plus connue est probablement l'**easing**.

Diawi : Diawi est un outil en ligne permettant aux développeurs d'installer une application sur téléphone ou tout autre appareil sans avoir à passer par les stores.

Facebook Developer : Facebook Developer permet de publier une application via Facebook, ainsi que l'accès à de nombreuses fonctionnalités liées à l'application comme la diffusion de publicité ou l'organisation de diffusions en direct.

Fastlane : Fastlane est une plateforme offrant une solution d'automatisation des étapes de publication d'une application mobile. Par exemple, Fastlane permet d'automatiser la construction d'exécutables, la réalisation de captures d'écran de l'application, et le déploiement en bêta ou sur les magasins.

Github : Github est un service cloud qui fait office de gestionnaire de versions des projets. Il permet aux développeurs notamment de travailler sur un même projet sans risquer d'écraser le travail d'un collègue et de le perdre (le travail) à tout jamais.

Github Actions : Github Actions permet d'exécuter des scripts automatiquement lorsque certains événements se produisent dans un répertoire sur Github. Cela permet par exemple la construction automatique d'exécutable, la réalisation de tests unitaires ou encore le déploiement d'application.

Google Meet : Google Meet est un outil de vidéo-conférence. Il permet à plusieurs collaborateurs de participer à des réunions professionnelles à distance, tout en intégrant un système de planification régulière ou non, ainsi que la possibilité d'avertir les collaborateurs du déroulement prochain d'une réunion par l'intermédiaire de rappels.

JetBrains Rider : est un logiciel d'édition de texte, particulièrement adapté aux fichiers de code, et encore plus particulièrement aux projets, comme c'est le cas des projets Unity. Il permet entre autres la détection d'erreurs de syntaxe pour éviter de découvrir l'absence d'un point-virgule juste avant la compilation, de créer automatiquement le template d'une classe d'après ses héritages, ou encore de vérifier l'accessibilité de classes et méthodes dans un fichier.

Kanban : Un Kanban est un outil de planification. Il permet de diviser un projet en plusieurs tâches, voir même les tâches en sous-tâches, et d'attribuer chacune de ces tâches à un membre du projet. On peut aussi ranger ces tâches de différentes façons, par ordre d'importance, par ordre chronologique ou par dépendances par exemple.

Notion : Notion est une application spécialisée dans la prise de notes, l'organisation et la gestion de contenu écrit collaboratif. Notion permet l'organisation d'idées pour les brainstormings, la création de tableaux Kanban, la diffusion d'information et de documentation. Elle permet également de gérer des bases de données.

Slack : Slack est une plateforme de communication collaborative. Elle permet à une équipe de communiquer par messages instantanés sur différents canaux et en messages privés, et d'effectuer de la gestion de projet.

Sourcetree : Sourcetree est une application permettant la gestion de repositories sur Github, facile la gestion des branches et la résolution de conflits entre les versions.

Unity : Unity est un moteur graphique et physique de jeu vidéo. C'est le moteur de jeu en C# le plus populaire au monde. C'est un moteur graphique car il permet à un développeur de disposer d'une interface graphique très simplement, ainsi que d'une scène pour y disposer des décors, des textures, des lumières et des shaders... C'est un moteur physique puisqu'il permet l'ajout de gravité, de forces en tout genre, de collisions et d'autres phénomènes physiques liés au mouvement sans que le développeur n'ait à écrire la moindre ligne de code. C'est donc un outil permettant de simplifier grandement le travail de toute personne travaillant sur le développement d'un jeu vidéo. Il permet également la création d'animations, si bien qu'il est même utilisé dans l'industrie du cinéma d'animation.

Unity Asset Store : L'Unity Asset Store est un magasin en ligne où les développeurs peuvent vendre ou acheter des scripts, des modèles, des textures ou quoi que ce soit d'autre pouvant servir au développement d'un jeu vidéo réunis dans un package. On peut y trouver par exemple de quoi implémenter facilement un générateur procédural de forêts, ou un script qui rend tous les objets qui lui sont associés mous comme un flan (un softBody).

Unity Clouds : Unity Clouds est un système de gestion des projets Unity en ligne. Il est notamment utilisé pour développer automatiquement des versions exécutables des jeux et pour réaliser des tests unitaires sur les projets.

Voodoo Academy : Voodoo Academy est un site web créé par la société d'édition de jeux vidéo Voodoo. Il permet aux développeurs hypercasual des studios édités de suivre de nombreux tutoriels pour se former au domaine de l'hypercasual, ainsi que de suivre des vidéo-conférences et diffusions en direct organisées par l'éditeur.

IX. Bibliographie

- Site officiel de Blue Monkey Studio
<https://bluemonkey.studio/>
- Description de Blue Monkey Studio sur Societe.com
<https://www.societe.com/societe/blue-monkey-studio-891268880.html>
- Documentation d'Unity
<https://docs.unity3d.com/Manual/index.html>
- Le site de Voodoo, la Voodoo Academy
Inutile de partager un lien, l'accès à ce site est confidentiel
- Documentation de Github Actions
<https://docs.github.com/en/actions>
- Documentation de Fastlane
<https://docs.fastlane.tools/>
- Documentation de Unity Clouds
<https://docs.unity3d.com/Manual/UnityCloudBuild.html>
- Documentation de l'API Slack
<https://api.slack.com/>
- Diawi
<https://www.diawi.com/>
- L'Unity Asset Store
<https://assetstore.unity.com/>
- Documentation de l'API Apple Store Connect
<https://developer.apple.com/documentation/appstoreconnectapi>

X. Annexe

Annexe1 : évaluation des compétences (tirée de la plateforme Horizon).

▼ Maîtrise des domaines scientifiques et techniques *

Capacité d'analyse / Compréhension des problèmes *
☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

Mise en oeuvre de ses connaissances *
☐ Très bien/Excellent ☒ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

Aptitude à acquérir de nouvelles connaissances *
☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

▼ Maîtrise des méthodes et des outils de l'ingénieur *

Méthodologie / organisation du travail, gestion de projet *
☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

Synthèse et communication des résultats, maîtrise des outils de communication *
☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

▼ Conduite de l'action et prise de décision *

Réalisation des objectifs - Qualité du travail réalisé *
☐ Très bien/Excellent ☒ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

Autonomie - initiative / créativité / ouverture d'esprit *
☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

▽ Intégration dans une organisation et capacité d'animation *

Capacité à s'intégrer dans une équipe *

☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

Communication sur ses activités et capacité à rendre compte *

☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

Prise en compte des enjeux métiers et économiques - Respect des procédures (qualité, sécurité, santé...) *

☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

▽ Respect des valeurs sociétales, sociales et environnementales *

Appropriation des valeurs, codes et culture de l'équipe et de l'organisation *

☐ Très bien/Excellent ☒ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

Attitude / assiduité / ponctualité *

☒ Très bien/Excellent ☐ Bien/Good ☐ Moyen/Average ☐ Insuffisant/Poor ☐ Sans objet/No subject

▽ Avant votre stage *

Vous êtes satisfait de l'accompagnement (offres, forums, conseils...) dont vous avez bénéficié pour trouver votre stage. *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☒ Plutôt d'accord ☐ Tout à fait d'accord

Les modalités d'évaluation du stage vous ont été clairement présentées avant le début du stage. *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☒ Plutôt d'accord ☐ Tout à fait d'accord

▽ Bilan du stage *

Vous êtes satisfait des conditions d'accueil sur votre lieu de stage. *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☐ Plutôt d'accord ☒ Tout à fait d'accord

Les conditions matérielles vous ont permis d'atteindre les objectifs du stage. *

☐ Pas du tout d'accord ☒ Plutôt pas d'accord ☐ Plutôt d'accord ☐ Tout à fait d'accord

Les missions qui vous ont été confiées sont en adéquation avec votre formation. *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☒ Plutôt d'accord ☐ Tout à fait d'accord

Vous êtes satisfait de l'encadrement dont vous avez bénéficié au sein de l'organisme d'accueil. *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☐ Plutôt d'accord ☒ Tout à fait d'accord

Avez-vous été informés par l'organisme d'accueil des conditions de sécurité? *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☐ Plutôt d'accord ☒ Tout à fait d'accord

▽ Compétences *

Ce stage vous a permis de prendre conscience de vos compétences. *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☐ Plutôt d'accord ☒ Tout à fait d'accord

Ce stage vous a permis de progresser dans la construction de votre projet personnel et professionnel. *

☐ Pas du tout d'accord ☐ Plutôt pas d'accord ☐ Plutôt d'accord ☒ Tout à fait d'accord

ge vous a permis de progresser dans la construction de votre projet personnel et professionnel.

▽ Appréciation globale *

Conseilleriez-vous cette entreprise pour un stage? *

☐ Non

☒ Oui