UCLouvain

epl

École polytechnique de Louvain

Jeu sérieux en ligne pour une méthode de développement agile

Auteur: **Guillaume Bellon**
Promoteur: **Manuel Kolp**
Lecteurs: **Ghazaleh Aghakhani, Kim Mens**
Année académique 2022–2023
Master [120] : ingénieur civil en informatique

Remerciements

J'aimerais tout d'abord remercier mon promoteur, le Professeur Manuel Kolp, pour son temps et ses conseils précieux.
J'aimerais aussi remercier ma maman pour son soutien sans faille et sans qui ce mémoire ne serait ce qu'il est aujourd'hui.
Enfin j'aimerais remercier ma sœur pour son aide sur les différents visuels présents dans l'application et ainsi que toutes les personnes qui auront testé le jeu lors de son développement.
Merci à vous tous.

Résumé

L'essor continu des méthodes agiles dans le domaine du développement logiciel témoigne de leur pertinence croissante pour répondre aux défis de la gestion de projets complexes. Il devient impératif pour les professionnels du secteur actuels ou en devenir de maîtriser ces approches afin d'améliorer l'efficacité, la collaboration et la qualité des produits développés. Cependant, le côté pratique inhérent de ces méthodes peut poser des défis en termes d'apprentissage et de diffusion efficaces des connaissances.

Le présent mémoire se propose de relever ce défi en explorant les possibilités offertes par les Serious Games comme méthode d'apprentissage des méthodes agiles. Ces jeux sérieux, en combinant ludicité et pédagogie, offrent un environnement interactif pour enseigner et appliquer des concepts clés de manière pratique et engageante.

En analysant les fondements des méthodes agiles, en examinant les principes pédagogiques des Serious Games et en concevant un logiciel pratique basé sur le jeu de plateau Scrumble, ce mémoire propose une méthode ludique et interactive pour aborder les méthodes agiles et fournit aux apprenants les outils nécessaires pour les appréhender.

Table des matières

Introduction

L'avènement de l'ère moderne a profondément remodelé la manière dont les interactions humaines s'entremêlent avec l'environnement qui les entoure. Cette dynamique a notamment impacté les processus de gestion et de travail à travers divers secteurs. Dans cet écosystème en perpétuelle mutation, les méthodes agiles ont émergé comme une réponse essentielle aux enjeux posés par la complexité croissante des projets contemporains et il est à présent indispensable de maîtriser ces méthodes de gestion de projets.

Objectif

L'objectif de ce mémoire sera de fournir un logiciel basé sur le concept des Serious Games et permettant de réaliser des parties du jeu de plateau déjà existant Scrumble afin de promouvoir la méthode Scrum inspirée du Manifeste Agile. Cette application aura donc avant tout un but pédagogique basé sur l'apprentissage par le jeu dans une optique de familiariser les utilisateurs avec le concept de la méthodologie Agile.

Plan du mémoire

Il vous sera tout d'abord présentée une analyse du contexte dans lequel se positionne ce travail en définissant les concepts dont il est question, ainsi que leurs intérêts et les enjeux auxquels ils se rapportent. Vous y trouverez aussi une explication quant aux motivations personnelles et publiques qui justifient la pertinence de ce sujet.

Ensuite, une étude de cas du jeu de plateau **Scrumble** vous sera présentée. Le jeu et son fonctionnement seront décrits afin d'en extraire les fonctionnalités intrinsèques au jeu d'origine, suivis d'une analyse des besoins supplémentaires propres à l'exportation numérique du jeu. L'identification desdites fonctionnalités aura pour but d'établir un cahier des charges des comportements à retrouver dans le produit final.

Une fois les besoins de l'application identifiés, il sera décrit les choix d'implémentations qui ont été effectués ainsi que les raisons qui ont poussé à l'utilisation de certaines technologies plutôt que d'autres.

Viendra alors la partie liée à l'architecture même de l'implémentation du projet. C'est dans cette partie qu'il vous sera décrit les différentes phases du jeu, l'ensemble des intervenants ainsi que leurs interactions menant au déroulement correct d'une partie.

Après l'architecture et l'implémentation d'une solution, vous trouverez une explication sur les différentes phases de tests qui ont été réalisées afin de conduire le produit final dans l'état où il se trouve actuellement.

Suivra à cela une série de pistes d'améliorations de l'application et une analyse critique des fonctionnalités pouvant être rajoutées à l'avenir, ainsi qu'une conclusion résumant l'ensemble des points qui auront été abordés durant ce travail.

Chapitre 1

Contexte

Avant de poursuivre, il est important de définir le contexte de ce travail de fin d'étude afin d'en comprendre les enjeux.

1.1 Serious Game

Les jeux sérieux, également connus sous le nom de "Serious Games", sont des applications interactives conçues avec l'intention d'éduquer, de former ou d'informer les utilisateurs tout en exploitant les caractéristiques ludiques inhérentes aux jeux[16]. Au fil des années, ces outils ont gagné en popularité en tant qu'approche innovante pour l'apprentissage, offrant une combinaison unique d'engagement, d'interactivité et d'immersion. Plusieurs études et recherches ont mis en évidence les nombreux bénéfices que les Serious Games apportent au processus d'apprentissage, et ce, dans divers domaines éducatifs.
En rajoutant une dimension interactive à l'apprentissage, on permet ainsi de se familiariser de manière plus efficace avec l'objectif du jeu.

1.1.1 Avantages

Voici quelques-uns des principaux avantages reconnus des Serious Games.

Engagement amélioré Les jeux sérieux captivent l'attention des apprenants en proposant des environnements interactifs et stimulants. L'élément ludique motive les utilisateurs à s'impliquer activement dans le processus d'apprentissage, ce qui peut améliorer la rétention des informations et favoriser une meilleure compréhension des concepts.[14] [24]

Apprentissage actif

Contrairement aux méthodes traditionnelles d'enseignement passif, les Serious Games encouragent l'apprentissage actif. Les apprenants sont incités à prendre des décisions, à résoudre des problèmes et à interagir avec des scénarios réalistes, ce qui favorise une meilleure compréhension et une application pratique des connaissances.[24] [12]

Répétition et pratique

Les jeux sérieux offrent souvent la possibilité de répéter des scénarios, de résoudre des défis et de s'exercer de manière récurrente. Cette répétition permet aux apprenants de consolider leurs compétences et de perfectionner leurs connaissances de manière progressive[16].

Environnements sécurisés

Certains domaines d'apprentissage nécessitent la pratique dans des environnements sécurisés, où l'échec n'entraîne pas de conséquences graves. Les Serious Games fournissent ces environnements virtuels, permettant aux apprenants de tester différentes approches et de tirer des leçons de leurs erreurs.[24]

Application contextuelle

Les Serious Games peuvent simuler des situations réelles, offrant aux apprenants une expérience concrète dans un environnement contrôlé. Cela permet une meilleure compréhension de la manière dont les connaissances peuvent être appliquées dans des contextes pratiques.

Apprentissage collaboratif

Certains jeux sérieux favorisent la collaboration et la compétition entre les apprenants, ce qui encourage l'apprentissage entre pairs et renforce les compétences sociales.[24]

1.1.2 Pertinence

L'intérêt des jeux sérieux est alors tout trouvé. C'est une manière d'apprendre efficace et interactive qui permet de se familiariser avec des concepts importants et dans un cadre sérieux. L'apprentissage est la principale fonction des serious games, que ce soit dans un cadre scolaire ou même dans un cadre professionnel. L'utilisation de l'informatique et du digital dans notre quotidien ne fait qu'augmenter, il est donc essentiel d'orienter l'apprentissage sur des supports qui conviennent à chacun, et surtout qui leur plaisent. Un événement récent va même accélérer ce besoin de revoir les méthodes d'apprentissage traditionnelles : la pandémie de COVID-19. Bon nombre de personnes se sont retrouvées dans des conditions d'apprentissage tout à fait exceptionnelles. Continuer de faire parvenir des savoirs efficacement, malgré que chacun se retrouve chez soi, est un challenge et l'enjeu pour trouver de nouvelles méthodes d'apprentissage est donc urgent et proposer des formations ludiques et en ligne peut tout à fait changer la donne.

1.1.3 Quelques jeux connus

Foldit

Développé par plusieurs scientifiques universitaires, Foldit est unique en son genre puisqu'il propose, au moyen du jeu, de contribuer à la recherche avancée sur la santé humaine, à la bio-ingénierie de pointe et au fonctionnement interne de la biologie. Le but du jeu ? concevoir de nouvelles protéines, molécules ou encore résoudre les problèmes de structures de protéines.
Illustration d'une partie de Foldit à la figure 1.1.
(Vous trouverez plus d'informations sur Foldit et son fonctionnement sur leur site Internet : `www.fold.it`).

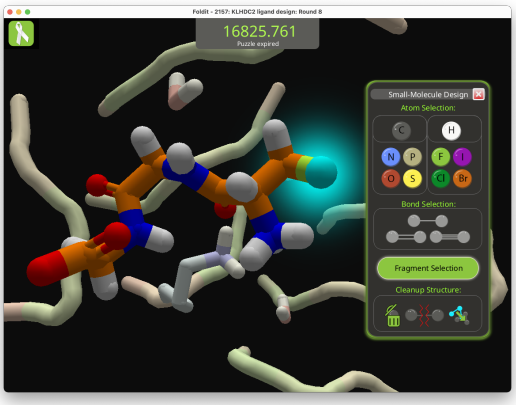


FIGURE 1.1 – Conception de molécule dans une partie de Foldit

Kanbanboard

Le jeu représente un système Kanban simulant une situation de développement de logiciel avec un product backlog représentant des user stories à instancier. Il permet de mettre en avant les bienfaits et avantages de la méthode Kanban, très utilisée dans un cadre de développement Agile, tout en conservant le côté ludique d'un jeu de plateau. On y retrouve bien les bienfaits des Serious Games décrits précédemment.
Illustration d'une partie de Kanbanboard à la figure 1.2.
(Vous pouvez faire une partie de Kanbanboard sur `www.kanbanboardgame.com`).

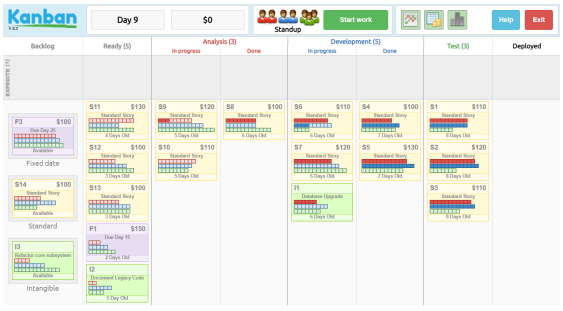


FIGURE 1.2 – Partie de Kanbanboard

Autres

On peut également citer les jeux suivants comme faisant partie de la famille des Serious Games :

- Eco : Un jeu dans lequel les joueurs doivent collaborer pour construire une civilisation dans un monde où tout ce qu'ils font affecte l'environnement (`www.play.eco`).
- World Without Oil : Ce jeu nous plonge dans un monde sans ressource pétrolière et nous invite donc à travailler en équipe pour imaginer des solutions à cette problématique (`www.writerguy.com/wwo/metahome.htm`).
- Play Spent : un jeu de simulation où les joueurs doivent prendre des décisions financières difficiles tout en essayant de survivre avec un budget limité. Cela souligne l'importance de la planification financière, de la gestion de l'argent et de la compréhension des priorités (`www.playspent.org`).

1.2 Qu'est-ce qu'une méthode de développement ?

Le deuxième terme à bien définir est *méthode de développement*.
Une méthode de développement est une manière d'aborder un processus de développement.
Initialement apparues dans le cadre de l'ingénierie logicielle, elles se sont rapidement popularisées dans de nombreux domaines de par leur efficacité. L'idée de ces méthodes de développement est une approche globale qui définit les principes, les pratiques, les lignes directrices et les étapes à suivre pour planifier, concevoir, construire et livrer un produit logiciel. Les méthodes de développement fournissent un cadre pour organiser le travail d'une équipe de développement et s'assurer que les processus sont cohérents, efficaces et de haute qualité. Les méthodes de développement et les cycles de développement sont interconnectés. Les méthodes de développement définissent comment les activités doivent être effectuées à chaque phase du cycle de développement, en fournissant des directives spécifiques pour gérer les exigences, la conception, l'implémentation, les tests, etc. Les cycles de développement, tels que le modèle en cascade ou le modèle en V, décrivent la séquence globale des étapes du processus de développement.

Lorsque nous appliquons une méthode de développement, nous suivons généralement un cycle de développement particulier en organisant les étapes du processus conformément aux principes et aux pratiques de la méthode. Par exemple, dans le cas de Scrum (méthode), nous organisons nos itérations en suivant un modèle itératif (cycle) en fonction des principes Agiles. Il convient donc d'également présenter les principaux cycles de développement afin de comprendre au mieux les concepts dont il est question.

1.2.1 Cycles de développement

Vous trouverez dans cette section une présentation des principaux cycles de développement : le modèle en cascade, le cycle en V et le modèle itératif.

Modèle en cascade

Le premier cycle de développement que l'on verra ici est le modèle en cascade. Initialement imaginé par Winston W. Royce en 1970[29], le modèle en cascade se base sur une suite de livrables et de phases. Analysons les différentes étapes de

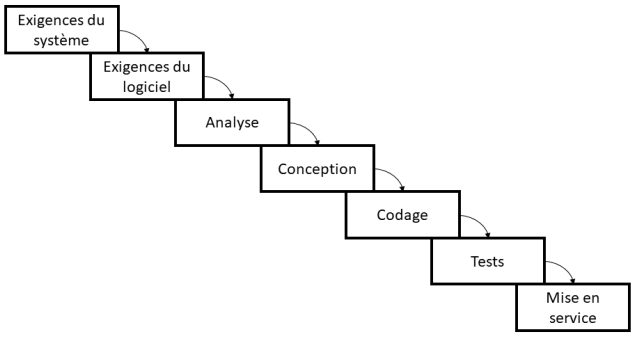


FIGURE 1.3 – Étapes de développement génériques du modèle en cascade

développement du modèle en cascade présenté à la figure 1.3.

1. Définition des exigences : Il est important de connaître les différentes exigences relatives au projet en lui-même. C'est dans cette phase qu'on va

définir les objectifs à atteindre, donner les spécifications attendues, poser les fonctionnalités désirées et connaître les contraintes liées au projet. C'est aussi et surtout dans cette phase qu'on crée le lien avec le client car c'est lui qui donne ses exigences vis-à-vis de la solution qui doit être livrée. C'est donc l'étape où on récupère toutes les informations nécessaires à l'élaboration du projet.

2. Analyse : Ensuite vient l'analyse. Cette étape consiste à formaliser les exigences voulues, les contraintes et autres fonctionnalités décrites lors de la phase de définition des exigences. De cette formalisation apparaîtra alors le cahier des charges du projet afin que tout le monde puisse se baser sur une formulation commune.

3. Conception : L'étape de la conception vise à imaginer l'architecture du projet, savoir comment il se comportera et concevoir le squelette du code. Le but ici est de préparer le terrain pour l'étape qui suit, le codage, afin que chacun puisse développer une fonctionnalité spécifique tout en restant cohérent avec l'architecture globale et le schéma commun.

4. Codage : Le codage représente le moment où le projet est réellement implémenté. Après avoir recueilli les différentes exigences, les avoir formalisées et avoir imaginé un squelette de la solution, il faut concrétiser cela en développant le projet. À la fin de cette étape, toute l'implémentation du projet doit être terminée.

5. Tests : Après le codage vient la phase de tests. C'est dans cette phase qu'on vérifie si toutes les fonctionnalités implémentées sont bien fonctionnelles, mais aussi et surtout, c'est là qu'on vérifie si la solution proposée correspond bien aux exigences posées dans le cahier des charges. Tous les aspects sont inspectés afin d'être sûr que tout fonctionne en adéquation avec les objectifs initiaux.

6. Mise en service : La dernière étape est la mise en service du logiciel. Après tout ce processus de développement et de test, c'est là qu'on retrouve le livrable final, prêt à être mis en service.

Critique Ce modèle a malheureusement quelques défauts. En effet, selon le modèle original, il est impossible de "remonter le courant" et de revenir à une étape précédente, ce qui implique que chaque étape terminée ne peut être reprise ou améliorée.[28] Winston W. Royce exprimait lui-même les limites de son modèle, le jugeant trop risqué envers les possibles erreurs se glissant au fil des étapes.[29] Il imagina alors un autre modèle, tenant plus compte des possibles erreurs pouvant survenir lors des étapes de développement. Ce modèle représenté à la figure 1.4 montre bien la possibilité de revenir d'une étape en arrière si un problème survient, permettant ainsi de gérer les éventuelles modifications.

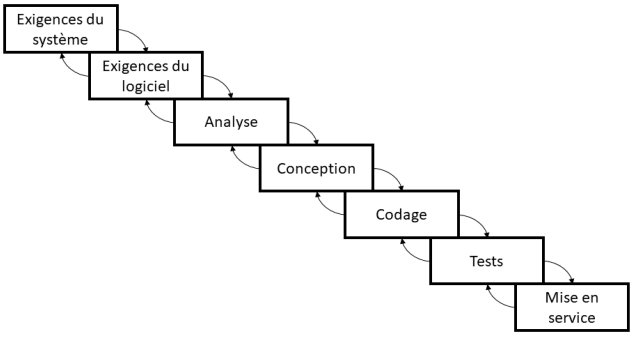


FIGURE 1.4 – Modèle en cascade amélioré

Cycle en V

Conçu dans les années 80, ce modèle de développement tire son origine du modèle en cascade. Il reprend la notion de phases successives descendantes présente dans ce dernier, mais y rajoute une deuxième phase, une phase ascendante, ce qui lui vaudra son nom de cycle en **V**. Cette phase ascendante remplace la phase de tests du modèle en cascade afin d'y apporter plus de rigueur. En effet, chaque étape descendante doit être validée par son étape de tests ascendante corrsepondante. De cette manière, on arrive à bien mieux gérer les éventuels "retours en arrière" puisque chaque étape doit être validée avant de passer à la suivante. Vous

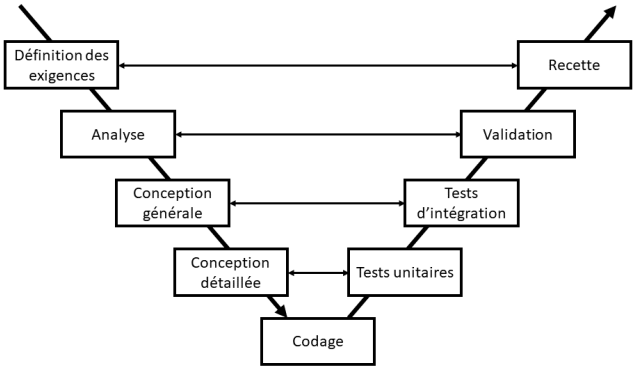


FIGURE 1.5 – Etapes de développement du cycle en V

trouverez à la figure 1.5 une représentation des différentes étapes présentes lors d'une conception de cycle en V. Étant basé sur le modèle en cascade, on retrouve similairement les mêmes étapes dans la phase descendante. La phase ascendante, quant à elle, est une amélioration de la phase de tests présente dans le modèle d'origine. Au lieu de ne faire qu'une phase de tests, on réalise une phase de tests en lien avec chaque phase de la pente descendante, ce qui permet de valider au mieux le logiciel et de vérifier que chaque étape respecte bien sa phase de conception. La partie descendante est donc plus une phase de développement théorique et de définition du projet en lui-même. La phase montante correspond à une série de tests vérifiant si la phase théorique correspondante est bien respectée et validée.
On remarquera tout de même la séparation de la phase d'architecture en deux phases de conception : la conception générale et la conception détaillée. La conception générale correspond à l'architecture globale du projet, à la manière dont chaque module va s'articuler avec les autres ; c'est la forme générale de la solution. La conception détaillée correspond, quant à elle, à une représentation des modules à proprement parler et à la manière dont ils réaliseront les fonctions exigées par la conception générale ; c'est l'architecture plus spécifique des parties du code.

Critique Le modèle en cycle V reprend les forces du modèle en cascade avec, notamment, ses phases successives permettant de structurer la continuité d'un

projet tout en l'améliorant. En rajoutant des phases de validation de chaque étape descendante, on s'assure du bon fonctionnement de chaque étape et on a un regard critique sur le développement. Cela permet d'apporter un regard sur les étapes précédentes, ce qui n'était pas tout à fait permis dans le modèle en cascade de base. Le modèle en cycle V hérite donc des avantages de son prédécesseur et parvient même à l'améliorer, mais cependant il hérite aussi de son principal point faible. On parle ici d'effet tunnel car, une fois le projet lancé, même s'il est basé sur des étapes bien précises, il est impossible de modifier fortement les étapes descendantes[22]. En effet, dans ce modèle de développement, si les attentes du client changent ou que les conditions initiales venaient à devenir obsolètes, le modèle du cycle en V ne permet pas ce genre de changements et tout le développement s'effondre. C'est cette rigueur d'exécution qui fait sa force, mais aussi sa principale faiblesse.

Modèle itératif

Le cycle de développement itératif est une approche qui consiste à diviser le processus de développement en itérations répétées, chaque itération produisant un résultat partiellement complet du produit final. Contrairement à une approche linéaire comme le modèle en cascade, où chaque phase est complétée avant de passer à la suivante, le cycle de développement itératif permet des retours constants, des ajustements et des améliorations à mesure que le produit prend forme. Les principaux aspects du cycle de développement itératif sont les suivants[28] :

- Itérations répétées : Le processus est découpé en plusieurs itérations. Chaque itération représente un cycle complet de planification, de conception, de mise en œuvre, de tests et de livraison.
- Développement progressif : Chaque itération produit une version partielle du produit final, ajoutant progressivement de nouvelles fonctionnalités et améliorations.
- Rétroaction continue : À la fin de chaque itération, une rétroaction est recueillie auprès des parties prenantes, des clients et des utilisateurs finaux. Cette rétroaction est utilisée pour guider les itérations suivantes.
- Adaptabilité : Le cycle itératif permet de s'adapter aux changements et aux évolutions des besoins tout au long du projet. Les ajustements peuvent être

apportés à chaque itération en fonction des nouvelles informations et des commentaires reçus.

— Réduction des risques : Le cycle de développement itératif permet de réduire les risques en identifiant et en résolvant les problèmes à un stade précoce du projet, au lieu d'attendre jusqu'à la fin.

— Livraisons fréquentes : Grâce aux itérations, des versions partielles fonctionnelles du produit sont livrées à intervalles réguliers, permettant aux clients de voir le progrès et de fournir des commentaires.

L'approche itérative est souvent associée aux méthodes de développement agiles, comme Scrum et Extreme Programming (XP) qui seront définies plus bas. Cependant, il existe également d'autres approches de développement itératif qui ne sont pas nécessairement agiles.

L'avantage principal du cycle de développement itératif est sa flexibilité face aux changements, sa capacité à prendre en compte les retours des parties prenantes et à s'adapter aux besoins changeants. Cela en fait une approche adaptée aux projets où les exigences ne sont pas clairement définies dès le départ, ou lorsque les besoins évoluent rapidement.

Critique Bien que le cycle de développement itératif offre des avantages, lui non-plus n'est pas exempt de critiques. Sa nature itérative peut rendre la gestion complexe, exigeant une surveillance constante des itérations et des ajustements rapides. Cela peut entraîner des coûts supplémentaires en ressources et en temps pour gérer les itérations et les ajustements nécessaires. De plus, la visibilité initiale peut être limitée, compliquant la planification et la communication. Les ajustements fréquents peuvent également augmenter la complexité technique et créer des défis d'intégration et de qualité du code. La nécessité de mettre à jour fréquemment la documentation peut être source de confusion et de travail supplémentaire. De plus, les dépendances constantes vis-à-vis des parties prenantes peuvent être difficiles à maintenir sur toute la durée du projet.
Bien que l'approche itérative offre de la flexibilité, ces points sont des considérations importantes à prendre en compte lors de son adoption[28].

Autres

Il existe bon nombre d'autres cycles de développement. Nous citerons également les cycles suivants :

— Modèle en Y[9].
— Prince2[18].
— PMBOK[27].
— Cycle incrémental.
— Modèle en spirale.

1.2.2 Méthodes agiles

Les méthodes agiles représentent une famille de méthodes regroupant des objectifs similaires et privilégiant des valeurs communes, à savoir :

— Les individus et leurs interactions, plus que les processus et les outils
— Des logiciels opérationnels, plus qu'une documentation exhaustive
— La collaboration avec les clients, plus que la négociation contractuelle
— L'adaptation au changement, plus que le suivi d'un plan

Découleront de ces valeurs, décrites pour la première fois dans le Manifeste Agile paru en 2001[11], une série de principes sous-jacents :

1. Satisfaire le client.
2. Accueillir positivement le changement.
3. Délivrer fréquemment.
4. Travailler avec les utilisateurs/clients.
5. Travailler avec des personnes motivées.
6. Privilégier le dialogue comme moyen de communication.
7. Avoir un logiciel qui fonctionne.
8. Encourager un rythme soutenable.
9. Fournir une attention continue.
10. Garder une simplicité.
11. S'auto-organiser en tant qu'équipe.

12. Réfléchir et s'adapter.

Les méthodes issues de cette famille s'orientent souvent autour d'un cycle de développement itératif ou incrémental et se distinguent fortement des autres méthodes de développement via leurs valeurs.
Tout d'abord, elles mettent en avant la communication dans l'équipe et favorisent l'interaction des membres, choses qui étaient légèrement mises de côté lors de la conception suivant les modèles en cascade ou de cycle en V. Ici ce n'est plus le processus de conception du projet qui est mis en avant, mais plutôt la manière dont chacun avance.
Le but de cette méthodologie est avant tout de proposer un logiciel opérationnel, plutôt que de fournir une documentation exhaustive. Autant mettre un maximum d'effort dans la partie la plus importante, à savoir, le produit final en lui-même, le logiciel.
Changement important, ici les méthodes agiles mettent également un point d'honneur à d'abord mettre en avant les rapports avec le client. En effet, bon nombre de méthodes agiles apportent un rôle essentiel au propriétaire du logiciel au sein de leurs réunions. Le client n'est plus seulement celui qui donne ses attentes en début de projet, mais celui qui donne son avis sur les avancées et les directions que prend le projet. Il est un membre de la résolution du problème à part entière, là où, dans d'autres méthodologies, il jouait un rôle plutôt extérieur au développement du projet, ne participant qu'aux étapes initiales et finales.
Enfin, les méthodes agiles cherchent à faire fi de l'inconvénient principal des autres méthodes de développement, le fameux effet tunnel cité précédemment. Ici la priorité est mise sur l'adaptation et sur la réactivité de l'équipe face aux changements[4].

Chaque méthode agile s'articule autour de ces quatre valeurs principales, mais varie dans sa manière de les présenter et d'organiser ses phases de développement[5]. Aussi, il vous sera décrit plus bas quelques méthodes agiles importantes pour la suite.

Scrum

Développé par Ken Schwaber et Jeff Sutherland, *Scrum* est un framework basé sur la méthodologie Agile visant à aider au développement et à la maintenance d'un logiciel[3]. Rapidement, cette méthode s'est répandue dans le domaine professionnel comme une manière efficace d'aborder tout problème et non plus seulement la conception de logiciels, si bien qu'elle est utilisée dans de nombreux domaines[30]. Scrum porte à la base donc les mêmes valeurs que la famille dont il est issu, valoriser les interactions au sein de l'équipe, privilégier le résultat, savoir s'adapter à tout changement et intégrer le client dans le processus de réalisation du projet[4].

Scrum porte son nom de l'anglais et se traduit par "mêlée" en français. Ce nom fait allusion à la mêlée du rugby qui est une phase où toute l'équipe se regroupe. La mêlée est importante car elle permet de repartir sur de nouvelles bases et dans une nouvelle direction, ce qui correspond bien au caractère adaptatif des méthodes agiles.[15]

Cette méthode se base sur six principes[25][1] :

— Le contrôle du processus empirique : De cette manière, Scrum respecte la philosophie de base des méthodes agiles qui cherchent à améliorer la transparence, l'inspection et l'adaptation d'une équipe lors de la réalisation d'un projet.

— L'auto-organisation de l'équipe : Ce principe est axé sur les travailleurs d'aujourd'hui qui produisent une valeur nettement supérieure lorsqu'ils sont auto-organisés, ce qui se traduit par une meilleure adhésion de l'équipe et une propriété partagée, ainsi que par un environnement innovant et créatif plus propice au développement.

— La collaboration : Le travail collaboratif est un principe fondamental de Scrum. En s'axant sur la prise de conscience, l'articulation et l'appropriation, il préconise la gestion de projet en tant que processus de création de valeur partagée, avec des équipes qui travaillent et interagissent ensemble pour apporter la plus grande valeur possible.

— Accorder de la priorité selon la valeur des tâches : Les méthodes agiles mettent la priorité sur le produit final et sur le logiciel, plus que sur la docu-

mentation. Avec ce principe, Scrum met en évidence la volonté de produire le résultat le plus vite possible. Il est important de produire les fonctionnalités avec la plus grande importance le plus vite possible, et ce, dès le tout début du projet.

— Calculer le temps nécessaire pour chaque tâche : La gestion du temps est très particulière avec Scrum. Il est important de déterminer la durée de chaque tâche afin de planifier au mieux le développement du projet. Cette manière de déterminer le temps permet donc de mieux gérer le planning du projet, mais aussi de son exécution en posant des unités de temps comme les sprints ou les daily's.

— Le développement itératif : La grande spécificité de Scrum, c'est avant tout son côté itératif. Afin de gérer au mieux les changements relatifs au coeur du projet, cette méthode se concentre sur une série d'objectifs, appelés "sprints". Chaque sprint représente une phase où un objectif est à atteindre dans un temps donné. Une fois le sprint terminé, on réévalue le projet et on corrige le tir en cas de changement. On recommence ainsi un nouveau sprint. Nous reviendrons plus en détail sur le fonctionnement des sprints juste après. Ce développement itératif est donc important car il permet de tenir compte des changements qui émergent lorsque le projet est déjà lancé.

Sur base de ces principes, on perçoit bien la volonté d'implémenter une méthode agile. Pour que Scrum puisse être mis en place, il est primordial que l'équipe possède un Scrum Master, formé et au courant des bonnes pratiques du Manifeste Agile afin de gérer le bon fonctionnement des sprints, daily et autres phases d'un développement Scrum [31].

Fonctionnement Un développement de projet suivant la méthode Scrum se fait en plusieurs étapes reprises à la figure 1.6.

La première des différentes étapes représentées à la figure 1.6, est la création des histoires d'utilisateur, ou user stories en anglais (que l'on abrégera ensuite par **US**), et donc l'expression des exigences quant au logiciel qu'on veut produire. C'est une étape importante, qu'on retrouvait notamment dans les modèles en cascade et de cycle en V, car c'est là qu'on définit tout ce que doit pouvoir faire l'application. Lors de cette étape, on améliore la vision du projet qu'a l'équipe et on pose toutes

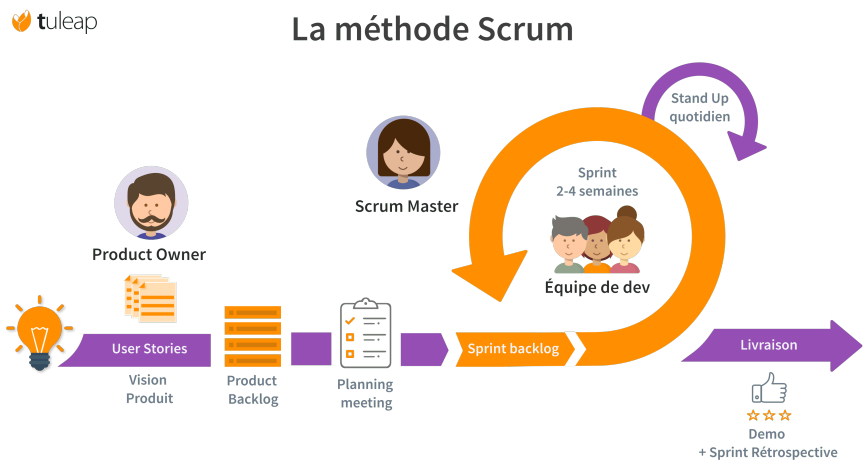


FIGURE 1.6 – Représentation des différentes étapes d'une méthode Scrum[2]

les fonctionnalités que doit avoir le logiciel.

Les US sont généralement fournies par le Product Owner, le client et propriétaire du produit final et correspondent avant tout à ses attentes quant à l'application, mais l'équipe doit bien entendu comprendre ce dont il est question dans les US.
De ces user stories apparaît l'élaboration d'un *Product Backlog* (ou carnet de commande en français). C'est dans ce carnet qu'on va regrouper toutes les user stories. On va également déterminer l'importance des fonctionnalités et leur durée par des pratiques telles que le planning poker [1].

À cela, suit le début du premier sprint. Les sprints, comme définis plus tôt, sont les unités temporelles de la méthode Scrum. L'équipe de développement travaille sur une petite quantité définie de US présentes dans le Product Backlog durant la totalité d'un sprint, qui forment le Sprint Backlog (ou carnet de sprint). L'intérêt du sprint est justement de diviser la totalité du travail en plusieurs itérations de développement, afin de mieux gérer les aléas et variations des US.

1. Le planning poker est une pratique où chaque membre d'une équipe estime, avec des cartes, la durée que prendra une étape de la réalisation du projet. De cette manière, tous les membres de l'équipe peuvent connaître les différents points de vue d'une fonctionnalité, débattre et surtout s'accorder sur la durée de ladite fonctionnalité[10].

Chaque jour d'un sprint, l'équipe de développement, un représentant client et un Scrum Master se réunissent afin de communiquer sur le projet lors de ce qui s'appelle un "Daily". Chacun doit prendre la parole durant cette phase de Daily afin de répondre aux trois questions suivantes :

— *Qu'ai-je fait hier ?*
— *Que vais-je faire aujourd'hui ?*
— *Quels obstacles ai-je rencontrés ?*

De cette manière, toute l'équipe peut avoir une meilleure vision globale de la quantité de travail qui a déjà été effectuée, ce qui est en cours, ainsi que communiquer des obstacles dès qu'ils apparaissent. Les phases de Daily ne sont pas les moments où l'on résout les problèmes et ne doivent pas dépasser plus de 15 minutes.

À la fin d'un sprint, on réalise une rétrospective du sprint effectué afin de tirer des conclusions quant à la manière de travailler, si elle est efficace ou non, ou encore quant à la quantité de US prévues pour le sprint. De cette manière, on agit rétroactivement sur les sprints suivants. On délivre les fonctionnalités implémentées et testées, on crée un nouveau Sprint Backlog en choisissant parmi les US toujours présentes dans le Product Backlog et on commence alors un nouveau sprint. On réitère alors autant de sprints que nécessaire afin de terminer toutes les user stories fournies par le Product Owner et de fournir toutes les fonctionnalités demandées[7].

Kanban

Issu du japonais, *Kanban* peut se traduire par "tableau d'affichage" et s'oriente plus sur une manière de représenter l'évolution du travail. Cette méthode consiste à diviser la totalité des fonctionnalités à implémenter en 3 principales catégories : **Todo**, à faire ; **In Progress**, en cours ; et **Done**, terminé. Chaque user story est alors, au début du projet, placée dans la colonne Todo et se déplacera dans le tableau en fonction de son état d'avancement. Il est alors plus facile pour l'équipe de visualiser le flux de travail du projet en chaque instant. La méthode Kanban implique aussi une limitation du nombre de US autorisées dans la colonne In progress et ses sous-catégories afin que le nombre de tâches actives reste maîtrisable.

Tout cela se réalise en tenant compte des effectifs dans l'équipe, de sa capacité disponible et du potentiel d'accroissement du rythme de livraison. En adaptant les besoins en ressources à la demande et en optimisant la gestion des blocages au niveau du système, cette stratégie vise à adapter au mieux le travail fourni sur les différentes US choisies. Elle représente à la fois le flux de travail et le travail réel impliqué dans le processus.
L'idée est donc de permettre de suivre plus aisément l'évolution du projet tout en le gardant à portée de main et en adaptant la quantité de tâches réalisées en fonction des capacités de l'équipe, ce qui se traduit par une optimisation des différentes phases de vie d'un projet[20].

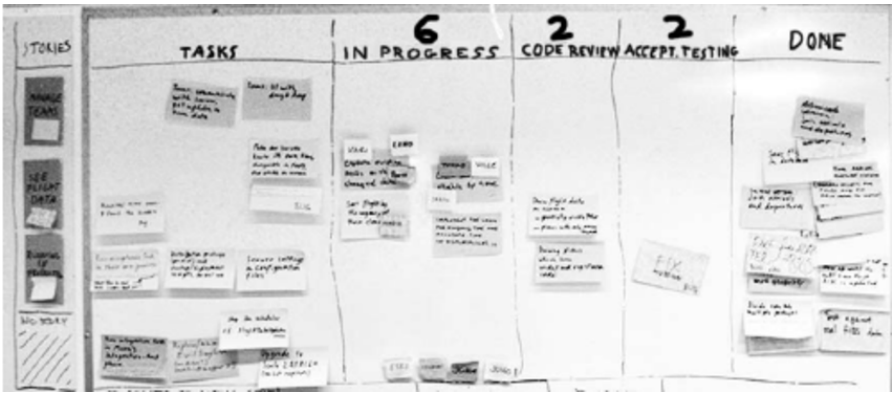


FIGURE 1.7 – Exemple d'une implémentation du tableau Kanban utilisé dans un projet de développement logiciel[20].

De la même manière que Scrum, Kanban propose aussi des réunions récurrentes appelées *cadences*. Ces événements réguliers permettent d'établir un flux de travail prévisible et facilitent l'amélioration continue. Plusieurs types de cadence se déroulent à des fréquences différentes représentées à la figure 1.8[7].

Extreme Programming

Extreme Programming ou *programmation extrême* en français, aussi appelée XP, est également une méthode de développement basée sur le Manifeste Agile, créée en 1999 par Kent Beck, Ward Cunningham et Ron Jeffries avec la parution du livre *Extreme Programming Explained*. Elle se distingue des précédentes méthodes

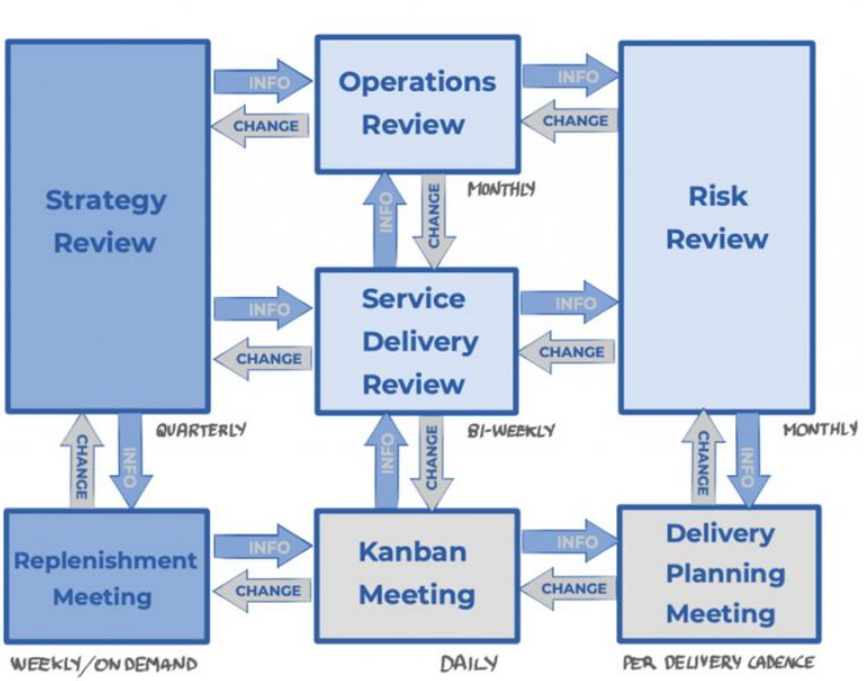


FIGURE 1.8 – Représentation des différentes cadences Kanban[6].

en mettant fortement l'accent sur une excellente application des techniques de programmation, une communication claire et le travail d'équipe. Selon sa définition originelle[8], XP inclut :

1. Une philosophie de développement de logiciels basée sur les valeurs de la communication, du feedback, de la simplicité, du courage et du respect.
2. Un ensemble de pratiques dont l'utilité a été démontrée pour améliorer le développement de logiciels. Les pratiques se complètent les unes les autres, amplifiant leurs effets. Elles sont choisies comme expression des valeurs.
3. Un ensemble de principes complémentaires, de techniques intellectuelles pour mettre des valeurs en pratique, utiles lorsqu'il n'existe pas de mode d'emploi pour un problème particulier.
4. Une communauté qui partage ces valeurs et bon nombre de ces pratiques.

Les valeurs décrites plus tôt au premier point sont essentielles pour comprendre la dynamique apportée par XP et seront donc expliquées un peu plus en détail[8].

La communication L'importance de la communication ouverte et continue est fondamentale dans XP. Une communication fluide entre les membres de l'équipe de développement, les clients et les parties prenantes garantit que les exigences sont claires et bien comprises.
Quelques pratiques liées :

— Programmation en binôme : Deux développeurs travaillent ensemble sur le même code, favorisant l'échange d'idées et la résolution de problèmes en temps réel.
— Planning Poker : L'équipe collabore pour estimer la complexité des tâches et établir des priorités en utilisant des cartes à jouer, stimulant les discussions constructives.

Les retours Dans des cycles courts comme ceux d'XP, il est plus qu'important d'avoir des retours rapides sur plusieurs niveaux. Extreme Programming met l'accent sur la rétroaction constante pour s'assurer que le logiciel réponde aux besoins des utilisateurs et évolue en conséquence.
Quelques pratiques liées :

— Développement piloté par les tests (TDD) : Les tests unitaires sont écrits avant le code, garantissant que le code fonctionne correctement et est maintenable.
— Propriété collective du code : Tous les membres de l'équipe sont responsables du code, ce qui encourage les retours et la collaboration continue.

La simplicité XP encourage à rechercher des solutions simples plutôt que complexes. L'objectif est de créer du code propre, facile à comprendre et à entretenir.
Quelques pratiques liées :

— Petits délivrables : Le logiciel est livré sous forme de petites versions fréquentes, ce qui facilite la gestion des changements et réduit le risque d'erreurs majeures.
— Refactorisation : Le code est constamment amélioré et réorganisé pour éviter la dette technique et maintenir sa simplicité.

Le courage Il représente une dimension essentielle qui pousse les équipes à prendre des mesures audacieuses pour améliorer la qualité du code, la communication, la collaboration et l'efficacité globale du processus de développement. Le courage dans XP implique la volonté de sortir de sa zone de confort, de relever des défis et de prendre des décisions difficiles pour atteindre des résultats positifs.
Quelques pratiques liées :

- Intégration continue : Les changements sont intégrés fréquemment pour éviter les problèmes d'intégration à grande échelle, nécessitant du courage pour relever et résoudre les conflits rapidement.
- Client sur site : Un client ou un représentant du client est impliqué dans le processus de développement, ce qui nécessite le courage de traiter directement avec les utilisateurs finaux.

Le respect La valeur du respect dans l'Extreme Programming joue un rôle crucial dans la création d'un environnement de travail positif et collaboratif. Cette valeur vise à promouvoir le respect mutuel entre les membres de l'équipe de développement ainsi qu'entre l'équipe et les parties prenantes externes, telles que les clients et les utilisateurs finaux.
Quelques pratiques liées :

- Rythme soutenable : Les horaires de travail sont maintenus à un niveau soutenable pour prévenir l'épuisement, témoignant du respect envers le bien-être de l'équipe.
- Collaboration avec le client : La collaboration continue avec le client favorise la confiance et le respect mutuel.

Vous trouverez un plan des différentes phases de gestion d'un projet sous XP à la figure 1.9.

Autres

Beaucoup d'autres méthodes agiles ont vu le jour au fil du temps. Nous citerons également les méthodes de développement suivantes[4] :

- Crystal clear
- FDD (Feature-Driven Development)
- RUP (Rational Unified Process)

FIGURE 1.9 – Plannification et boucles de feedback dans Extreme Programming[13].

— DSDM (Dynamic Systems Development Method)

Critiques

Certains des signataires du Manifeste Agile émettent cependant certaines réserves quant à l'utilisation de la méthodologie Agile aujourd'hui. Ron Jeffreis, l'un d'eux, explique dans son article "Developers Should Abandon Agile"[21] que, pour lui, les méthodes agiles sont devenues très, voire trop, commerciales. Elles ne représentent plus certaines des valeurs initialement posées, à savoir entre autres le bien du développeur dans la conception du projet. Elles mettent trop en avant le bon fonctionnement du projet et le produit fini en délaissant les interactions et la place que chacun peut avoir dans ce processus de développement. Ron Jeffries n'indique pas que toute la méthodologie Agile est à abandonner, mais qu'il faudrait revenir à ses fondements car son utilisation a été corrompue par le côté commercial du développement. Agile est une bonne chose et il invite chacun à revoir les valeurs originales d'Agile, présentes dans le manifeste, plutôt que de l'abandonner à proprement parler.

Andy Hunt, également signataire du Manifeste Agile, tire les mêmes observations[19].

Pour lui, le modèle Agile ne fonctionne pas car, ce dernier qui se veut abstrait et prêt à embrasser le changement plutôt que de le craindre, n'est pas fait pour des personnes non-expérimentées. Il dit que les gens ne peuvent pas se rendre compte de ce qu'est réellement une méthode agile car ils ne l'ont pas encore assez expérimentée ou même comprise dans ses fondements. Il explique qu'il est normal pour un novice qui découvre un nouvel outil de se référer aux règles, de chercher à savoir exactement quoi faire si tel problème survient, alors que la méthodologie Agile se veut justement beaucoup plus abstraite vis-à-vis du concept d'analyse et d'adaptation. Ils ne voient pas la méthodologie Agile comme un courant s'adaptant aux situations et permettant d'atteindre les valeurs agiles de manière abstraite, mais plutôt comme des règles fondées et intransigeantes sans chercher à voir au-delà.

1.2.3 Avantages

L'intérêt des méthodes de développement reste avant tout d'apporter une certaine rigueur quant au cycle de vie d'un projet. En faisant de la sorte, on arrive à planifier toutes les étapes du développement et chacun a une meilleure vision de la réalisation du projet. Chaque méthode de développement s'articule en fonction des objectifs qu'elle vise. De cela découle qu'il n'existe pas une méthode meilleure que les autres, mais bien des méthodes plus spécialisées dans certaines conditions ou dans le but de mettre en priorité des aspects différents. Il est donc tout à fait important de connaître ces différentes manières de planifier la résolution d'un projet.

1.3 Objectif du mémoire

Maintenant que le contexte a été mis en place, il est temps de comprendre les différents enjeux que soulève ce thème et sa pertinence comme choix de travail de fin d'études.

1.3.1 Enjeux publics

Développer un Serious Game sur le thème des méthodes agiles présente des enjeux importants et des avantages significatifs pour le domaine professionnel. Les méthodes agiles, telles que Scrum et Kanban, sont de plus en plus adoptées dans diverses industries pour améliorer la gestion de projets et la collaboration. Créer un Serious Game axé sur cette méthodologie apporte plusieurs avantages, tant du point de vue de l'éducation que de l'efficacité opérationnelle.

Promotion de l'agilité

L'un des principaux enjeux est de promouvoir la compréhension et l'adoption des méthodes agiles. Le monde professionnel est souvent confronté à des défis complexes nécessitant une gestion efficace des projets. Un Serious Game permettrait de sensibiliser ses utilisateurs aux avantages de l'agilité, encourageant ainsi l'adoption de ces méthodes pour des résultats plus efficaces et une meilleure compréhension de ces concepts.

Amélioration des compétences collaboratives

Les méthodes agiles mettent l'accent sur la collaboration, la communication et le travail d'équipe. Un Serious Game offrirait une opportunité pratique d'améliorer les compétences collaboratives des utilisateurs. En simulant des scénarios de projet réalistes, le jeu encouragerait la prise de décision en groupe, l'échange d'idées et le partage d'informations, ce qui est essentiel pour relever des défis complexes.

Renforcement de la gestion de projets

Les méthodes agiles offrent des approches flexibles pour gérer des projets en constante évolution. Le développement d'un Serious Game sur ce thème fournirait aux intéressés et aux responsables de projet des compétences en matière de planification, de priorisation et d'adaptation rapide aux changements.

Sensibilisation aux avantages de l'innovation

Les méthodes agiles encouragent l'innovation continue en offrant un cadre pour tester rapidement de nouvelles idées. En développant un Serious Game, il serait envisageable de promouvoir une culture d'innovation au sein des équipes. Les participants pourraient expérimenter des solutions créatives, des itérations rapides et des améliorations continues, ce qui peut stimuler l'approche proactive dans la résolution des problèmes.

Réponse aux besoins changeants de la société

Le monde professionnel doit souvent s'adapter aux changements rapides dans la société. Les méthodes agiles offrent la souplesse nécessaire pour répondre à ces besoins changeants. Un Serious Game centré sur l'agilité fournirait une expérience pratique pour apprendre à gérer les exigences changeantes, à ajuster les priorités et à maintenir une approche axée sur les résultats.

En somme, développer un Serious Game sur le thème des méthodes agiles offre une opportunité précieuse d'améliorer les compétences, de promouvoir l'efficacité et de renforcer la capacité d'adaptation dans un contexte où les défis sont nombreux et diversifiés. Ce type d'initiative peut contribuer à moderniser les pratiques professionnelles en alignant les méthodes de travail sur de meilleures pratiques de gestion de projet et de collaboration.

1.3.2 Choix personnel

Ce thème m'a intéressé pour une raison principale : il met en avant l'importance de trouver de nouvelles méthodes d'apprentissage et de faire apprendre des concepts. Il m'a toujours paru essentiel de partager les connaissances que l'on acquiert. À quoi servent-elles si on ne peut les partager avec autrui ? Ce thème de mémoire est pour moi une occasion d'en apprendre plus sur une manière de transmettre de l'information, mais aussi un moyen de participer à cet enseignement et de permettre à d'autres de comprendre au mieux ce concept de méthodologie Agile.

1.3.3 Pourquoi Scrumble ?

Le jeu Scrumble (que je décrirai plus en détail au prochain chapitre) est déjà un Serious Game existant et visant à familiariser ses utilisateurs à la méthode Scrum, une des méthodes les plus populaires appliquant le Manifeste Agile à l'heure actuelle. Développer un logiciel permettant de réaliser une partie de Scrumble est donc un bon moyen de transmettre le fonctionnement et le principe de Scrum tout en y associant les avantages de l'apprentissage par Serious Game.

Chapitre 2

Étude de cas

Il est maintenant temps de s'intéresser à l'élaboration de la solution en tant que telle. Il sera tout d'abord nécessaire de comprendre le jeu de base Scrumble afin d'en tirer une liste de fonctionnalités clés qui doivent se retrouver dans le délivrable, mais il sera tout aussi important de réfléchir aux ajouts qu'il faudra y apporter afin de rendre une exportation numérique viable. La liste des caractéristiques désirées mènera à l'élaboration d'un cahier des charges.

2.1 Scrumble : le jeu de plateau

Créé en 2014 par Pyxis Technologies, Scrumble se base sur l'idée que, si la méthode agile Scrum vient en partie d'un jeu, alors pourquoi ne pas créer un jeu sur Scrum. Scrumble s'oriente donc comme un Serious Game dont le but est de familiariser ses joueurs aux concepts de Scrum en leur faisant prendre les différents rôles présents dans une équipe Scrum et en leur faisant travailler sur des US de tailles et valeurs différentes tout en les laissant s'organiser au mieux[26].

2.1.1 But du jeu

Au début du jeu, l'équipe reçoit une liste de 15 user stories parmi lesquelles sont répartis 45 points de valeur, des étoiles, et qui constitueront le Product Backlog. En travaillant sur une US, elle finit par être terminée et rapporte à l'équipe autant de points qu'elle possède d'étoiles. Le but du jeu est d'obtenir les 45 points. D'autres

objectifs secondaires sont également disponibles comme réduire la dette technique qui évoluera durant la partie, ou encore acquérir les étoiles le plus vite possible ou en réduisant le nombre de sprints nécessaires au maximum, mais la fin de partie intervient lorsque toutes les US sont terminées et déployées.

2.1.2 Rôles présents dans une partie

Dans une partie de Scrumble, on retrouve les différents rôles essentiels au bon fonctionnement de Scrum :

- L'équipe de développement : Elle représente la majeure partie des joueurs et travaillera sur l'accomplissement des US et la réduction de la dette technique. Les membres de l'équipe de développement sont tous égaux et s'auto-organisent en vue de fournir les délivrables prévus.
- Le Product Owner : Il représente le client, fournit le Product Backlog et s'occupe de guider la partie. Au cours de la partie, il agit comme décisionnaire et intermédiaire entre l'équipe de développement et les utilisateurs ou les ayants droit commanditant fictivement le projet.
- Le Scrum Master : Il est le maître du jeu, et animateur de la partie. Son rôle est de s'assurer que tout se déroule selon les règles du jeu, mais aussi que les valeurs de Scrum soient mises en avant tout au long de l'expérience. Il est donc préférable que ce rôle soit endossé par une personne bien au courant de la méthode Scrum ou encore un réel Scrum Master.

2.1.3 Déroulement d'une partie

Avant de commencer une partie à proprement parler, il est important, au préalable, de récupérer les US auprès du Product Owner. Des US préécrites sont fournies dans le jeu, mais il est tout à fait possible de créer sa propre liste de US. Chaque US se compose d'une description de la finalité désirée, d'une valeur que rapportera la US une fois terminée, d'une taille et d'une possible restriction, une US pouvant nécessiter de terminer une autre US au préalable. Une fois les US fournies, c'est au tour de l'équipe de développement de décider de la taille de chaque US. La taille représente la complexité d'une fonctionnalité à être réalisée et donc

la quantité de travail qu'elle nécessitera. La taille doit être choisie parmi les tailles suivantes : XS, très simple ; S, simple ; M, normale ; L, complexe et XL, très complexe. Une fois que toutes les US se sont vues attribuer une taille, la préphase est terminée et il est temps de se lancer dans la phase de jeu.

Tout comme la méthode Scrum, la partie va se diviser en sprints. Un sprint est ici composé, par ordre chronologique, d'une plannification, de 9 jours de développement, d'une revue du travail accompli ainsi que d'une rétrospective.

Plannification La plannification est le moment où l'équipe de développement choisit quelles seront les US issues du Product Backlog sur lesquelles elle travaillera durant le sprint en les rajoutant au Sprint Backlog. C'est à ce moment que le nombre de tâches à réaliser pour une US choisie se décide en suivant la formule suivante :

Nombre de membres de l'équipe de développement $\times$ Taille de la User Story choisie $\times$ Facteur de la dette technique courante.

Chaque taille de US correspond donc à un multiplicateur : '1' pour XS, '2' pour S, '3' pour M, '4' pour L et enfin '5' pour les US XL.
Le facteur de dette technique, lui, varie au cours de la partie en fonction de la valeur de la dette technique courante. Pour une dette technique comprise entre :

- $[0, 9]$, le facteur sera de $\times 3$
- $[10, 19]$, le facteur sera de $\times 4$
- $[20, 29]$, le facteur sera de $\times 6$
- $[30, 39]$, le facteur sera de $\times 9$
- $[40, 49]$, le facteur sera de $\times 12$

La dette technique par défaut au début d'une partie vaut 15 et le facteur sera donc de $\times 4$.
L'équipe de développement doit donc bien tenir compte de la dette technique courante afin de ne pas devoir réaliser trop de tâches lors du sprint, mais aussi de gérer au mieux cette dette afin de rendre les futures US plus simples à réaliser au fil des sprints.

Jours de développement C'est dans cette phase que les développeurs vont faire progresser les US. Chaque jour, chaque joueur décide de gagner des tâches à répartir sur les US présentes dans le Sprint Backlog, ou bien de réduire la dette technique courante. Le nombre de tâches à ajouter ou de dette à réduire est défini par un lancer de dé de la part du joueur.
Si le dé affiche un '1', le joueur peut choisir à nouveau entre augmenter les tâches, et réduire la dette technique et peut relancer le dé (une seule fois par tour).
Si le dé affiche un '6', le joueur doit piocher une carte problème et subir l'effet de ladite carte.
Au début de chaque jour, l'équipe de développement pioche également une carte 'Daily' et applique son effet. Les cartes piochées interviennent comme des aléas faisant varier la dette technique courante, le nombre de tâches ou encore en donnant des informations sur la méthode Scrum.
À la fin d'un jour, on passe au jour suivant jusqu'à la fin du $9^{\text{ème}}$ jour qui marquera la fin des jours de développement et le début de la phase de revue.

Revue La revue est le moment où les joueurs font l'état des US terminées, des tâches qu'il reste à apporter aux autres et où l'on procède au décompte des points accumulés. Par tranche de 5 tâches manquantes à une US non terminée, la dette technique courante augmente d'un point, le facteur de dette technique pouvant dès lors lui aussi varier. L'équipe de développement pioche autant de cartes 'Revue' que de US terminées, afin de faire varier l'état de la partie pour les sprints suivants, avant de passer à la phase suivante : la rétrospective.

Rétrospective C'est le moment qui caractérise la fin d'un sprint et où chaque joueur peut interagir pour indiquer ce qu'il a aimé ou moins lors du sprint, ce qu'il pense qu'il faudrait améliorer ou ce qui s'est bien passé. Chacun est encouragé à discuter afin de voir si des choses doivent être mises en place pour le sprint suivant. C'est aussi ce moment qui est le plus formateur du jeu car c'est là qu'est censée émerger une dynamique en corrélation avec les méthodes agiles. Elle dure entre une et deux minutes.

Après cette dernière phase, le sprint est terminé et, s'il reste encore des US

dans le Product Backlog ou dans le Sprint Backlog, un nouveau sprint commence sur une phase de plannification.

2.1.4 Objectif de Scrumble

L'objectif derrière une partie de Scrumble est donc avant tout de découvrir ou de se familiariser avec les méthodes agiles, en particulier avec Scrum, en permettant aux joueurs de se mettre dans une situation concrète. On y retrouve des valeurs agiles telles que l'importance de l'individu et de ses interactions durant le développement grâce aux phases de rétrospective, la collaboration avec le client puisque ce dernier se trouve en tant que joueur dans la partie et interagit avec l'équipe ou encore l'adaptation au changement plus que le suivi du plan grâce aux divers sprints et aux événements aléatoires générés par les cartes 'Daily', 'Problème' et 'Revue'.

2.2 Cahier des charges

Maintenant que le jeu est défini, nous allons en tirer les fonctionnalités qui doivent être présentes dans une version automatisée. Nous listerons donc les comportements nécessaires au fonctionnement basique du jeu, mais nous définirons aussi les fonctionnalités en rapport avec l'automatisation même et les spécificités propres au délivrable.

2.2.1 Fonctionnalités présentes dans le jeu original

Afin de se positionner dans une méthode de développement agile, toutes les fonctionnalités seront proposées sous forme de user story. L'ensemble des US représentant les fonctionnalités de base du jeu Scrumble sont représentées à la table 2.1.

2.2.2 Fonctionnalités à ajouter

À cela viennent s'ajouter quelques fonctions supplémentaires qui seront décrites plus bas. Ces fonctions ne sont pas là pour modifier le jeu de base, mais pour y apporter des fonctions supplémentaires n'altérant pas le fonctionnement ou apportant des informations quant au déroulement de la partie.

1. Fournir une taille par défaut aux US pré-intégrées : Il serait intéressant pour les joueurs de fournir une taille par défaut aux US pré-intégrées étant donné que la phase d'attribution des tailles d'US peut être plutôt longue. Cela permettrait de lancer directement une partie sans passer par cette phase tout en ayant une certaine cohérence dans les tailles des US par défaut.
2. Exporter un Burndown Chart : Il serait intéressant de pouvoir exporter un Burndown Chart en temps réel et basé sur les choix des joueurs afin de visualiser au mieux leur progression et leur apporter des informations supplémentaires lors de la revue de sprint.
3. Visualiser un Scrum Board : Il serait intéressant de visualiser au mieux l'avancée des joueurs en terme de US en leur représentant un tableau Scrum.
4. Changer la difficulté de la partie : Il serait utile pour les joueurs de pouvoir choisir leur difficulté afin de personnaliser leur partie en fonction de leurs expériences.
5. Changer la langue de la partie : Il serait important de proposer au moins une deuxième langue afin d'augmenter la portée du jeu auprès des utilisateurs.
6. Sauvegarder une preuve de complétion de la partie : Il devrait être possible de sauvegarder la preuve de complétion lors de l'écran final, afin que chacun puisse prouver sa participation au jeu.

Ce qui permet d'établir la liste d'US reprenant les fonctionnalités supplémentaires présentées à la table 2.2.

2.2.3 Spécificités du délivrable

Il est également nécessaire de se pencher sur les nécessités d'environnement d'une application et de s'intéresser à sa portabilité. Nous rajouterons donc les fonctionnalités présentes à la table 2.3 au cahier des charges.

ID	Description
US 1.1	En tant que joueur, je veux pouvoir jouer des parties allant de trois à neuf joueurs.
US 1.2	En tant que Product Owner, je veux pouvoir choisir une liste de US pré-intégrée.
US 1.3	En tant que Product Owner, je veux pouvoir créer mes propres US à utiliser dans une partie.
US 1.4	En tant que Product Owner, je veux pouvoir ajouter des restrictions à certaines US.
US 1.5	En tant que membre de l'équipe de développement, je veux avoir la possibilité de choisir la taille des US sélectionnées.
US 1.6	En tant que membre de l'équipe de développement, je veux pouvoir ajouter des US au nouveau sprint débutant.
US 1.7	En tant que joueur, je veux pouvoir suivre la valeur de la dette technique au cours de la partie.
US 1.8	En tant que joueur, je veux pouvoir suivre la quantité de tâches réalisées pour chaque US.
US 1.9	En tant que membre de l'équipe de développement, je veux choisir d'ajouter des tâches aux US ou de réduire la dette à chaque tour.
US 1.10	En tant que membre de l'équipe de développement, je veux pouvoir ajouter des tâches selon un lancer de dé.
US 1.11	En tant que membre de l'équipe de développement, je veux pouvoir diminuer la dette technique selon un lancer de dé.
US 1.12	En tant que membre de l'équipe de développement, je veux pouvoir relancer le dé si un '1' est obtenu au premier lancer.
US 1.13	En tant que membre de l'équipe de développement, je veux pouvoir piocher une carte 'Problème' si un '6' est obtenu à un lancer de dé.
US 1.14	En tant qu'équipe de développement, je veux piocher une carte 'Daily' au début d'un jour de développement, hormis le premier.
US 1.15	En tant que joueur, je veux pouvoir choisir de rajouter des US au Sprint Backlog si toutes les US déjà choisies sont terminées.
US 1.16	En tant que joueur, je veux pouvoir choisir de poursuivre le sprint en ne pouvant que réduire la dette technique à chaque tour une fois toutes les US du Sprint Backlog terminées.
US 1.17	En tant que joueur, je veux pouvoir accéder à la phase de revue directement après avoir terminé toutes les US du Sprint Backlog s'il n'y a plus aucune US présente dans le Product Backlog.
US 1.18	En tant que joueur, je veux pouvoir gagner la valeur des US déployées lors de la revue.
US 1.19	En tant qu'équipe de développement, je veux pouvoir piocher autant de cartes 'Revue' que de US déployées durant la phase de revue.
US 1.20	En tant que joueur, je veux voir la dette technique augmenter d'un point par tranche de cinq tâches non terminées pour chaque US.
US 1.21	En tant que Scrum Master, je veux avoir deux minutes de temps pour permettre des interactions avec les membres du jeu.
US 1.22	En tant que joueur, je veux pouvoir réitérer des sprints jusqu'au déploiement de chaque US.

TABLE 2.1 – User stories issues du jeu initial

ID	Description
US 2.1	En tant que membre de l'équipe de développement, je veux pouvoir lancer une partie sans devoir choisir des tailles pour les US sélectionnées.
US 2.2	En tant que joueur, je veux pouvoir exporter un Burndown Chart afin de suivre l'avancée du projet.
US 2.3	En tant que joueur, je veux pouvoir visualiser un Scrum Board afin de mieux visualiser l'ensemble des US du projet.
US 2.4	En tant que joueur, je veux pouvoir changer la difficulté de la partie.
US 2.5	En tant que joueur, je veux avoir la possibilité de choisir la langue dans laquelle se déroulera le jeu.
US 2.6	En tant que joueur, je veux pouvoir exporter une preuve de complétion de la partie.
US 2.7	En tant que joueur, je veux pouvoir visualiser le résumé de mon sprint à la phase de revue.

TABLE 2.2 – User stories des fonctionnalités supplémentaires

ID	Description
US 3.1	En tant qu'utilisateur, je veux pouvoir lancer le logiciel depuis un environnement Windows.
US 3.2	En tant qu'utilisateur, je veux pouvoir lancer le logiciel depuis un environnement MacOS.
US 3.3	En tant qu'utilisateur, je veux pouvoir lancer le logiciel depuis un environnement Linux.
US 3.4	En tant qu'utilisateur, je veux que le logiciel soit adapté au format d'écran 16/9 (1920p x 1080p).

TABLE 2.3 – User stories liées au délivrable

Chapitre 3

Choix d'implémentation

Lors de l'élaboration de la solution, il a fallu faire quelques choix d'implémentation tels que l'architecture globale, comment organiser le code et faire interagir ses composants ; l'environnement de programmation, quel programme utiliser pour l'implémentation de l'application ; ou encore la méthode de développement adoptée afin de progresser de manière efficace et suivie lors de la phase de développement.

3.1 Choix de l'architecture globale

Une partie de Scrumble étant assez linéaire, il a d'abord été envisagé une architecture en **appels et retours** afin de gérer les différentes étapes du jeu. Le programme ferait donc appel à des fonctions spécifiques en fonction de la phase du jeu dont il est question, qui appelleraient d'autres sous-fonctions, etc. pour réaliser les diverses actions requises au bon fonctionnement du jeu.
Après un premier essai, il s'est très vite avéré qu'une telle architecture n'apportait pas assez de lisibilité quant à l'état de la partie et aux divers appels de méthodes. Le code aurait semblé difficilement maintenable et son évolution aurait été compliquée de par son côté très monolithique.
Afin de pallier ces problèmes, une architecture pensée sur le concept de microservices a été appliquée. En dédiant certains services à des objets spécifiques, l'organisation générale s'en retrouvait beaucoup plus compréhensible et favorisait l'ajout de nouvelles fonctionnalités. C'est donc cette organisation générale qui a

été choisie pour l'élaboration de la solution.

3.2 Choix de l'environnement de développement

Lors du test d'architecture en appels et retours, il était important d'avoir accès à une rapidité de développement et une facilité d'expérimentation tout en ayant accès à des librairies spécifiques au développement de jeu. Le premier langage utilisé aura donc été **Python** en utilisant la bibliothèque **PyGame**, permettant un développement en temps réel et la rapidité de développement demandée.
Lorsque l'architecture en microservices s'est révélée plus pertinente, Python et PyGame se sont retrouvés très pauvres en terme de fonctionnalités spécifiques au développement de jeu. Il a donc fallu passer sur un autre environnement de programmation : Unity.
Unity est un moteur de jeu permettant de coder facilement des jeux orientés-objet de par ses nombreuses bibliothèques et modules, mais aussi par sa possibilité de créer des objets et d'y attacher des scripts de comportement. Le langage de développement des scripts des différents composants est en **C#** et leur intégration dans le moteur de jeu est directement gérée par Unity, ce qui rend plus aisée une architecture basée sur le principe de microservices. Unity permet également d'exporter l'application développée sur de multiples plateformes, ce qui est un plus aux vues des spécificités du délivrable définies précédemment. C'est donc sur cet environnement, et plus particulièrement sur la section 2D de Unity, que le jeu sera développé.

3.3 Choix de la méthode de développement

Développant un Serious Game visant à faire connaître et comprendre les bienfaits des méthodes agiles, il semblait intéressant d'appliquer le Manifeste Agile lors du développement dudit Serious Game. Une fois les fonctionnalités reformulées en userstories et intégrées dans le cahier des charges bien défini, il a été décidé d'organiser le travail en séquences de développement, choisissant à chaque phase les US qui seraient implémentées. À la manière des sprints de la méthode Scrum, les

fonctionnalités choisies seront implémentées et testées durant chaque phase, puis déployées afin de permettre une intégration continue. Cette manière de procéder permettra une grande flexibilité quant aux changements qui pourront se produire durant les phases de développement, et mener le projet à bien.

Chapitre 4

Architecture

Une fois le cahier des charges terminé et les choix d'implémentation effectués, il est temps de parler de l'architecture de l'application en elle-même. Dans ce chapitre, vous trouverez une description détaillée de chaque composant et des interactions qu'il a avec chacun ainsi que les états dans lesquels se trouve le jeu en chaque instant.

4.1 Architecture globale

La première chose à analyser est l'architecture globale de l'application et comment cette dernière va organiser les différentes phases du jeu.

4.1.1 Séparation d'une partie en quatre phases

En observant le déroulement complet d'une partie de Scrumble, il apparaît quatre phases principales, à savoir :

— Le menu principal
— L'attribution des tailles de US
— La création de US personnalisées
— La partie en elle-même

Chacune de ces phases est particulière et est chargée d'une mécanique spécifique du jeu fortement différente des autres. Unity proposant un système de "Scène" pour gérer des fonctions fondamentalement différentes, quatre scènes seront donc

créées, une pour chaque phase. Vous trouverez à la figure 4.1 un diagramme d'état représentant les différentes phases et transitions par lesquelles passera le jeu au cours d'une partie. L'architecture de chacune des phases sera décrite plus bas.

Cependant, malgré le fonctionnement individuel et autonome de chacune de ces

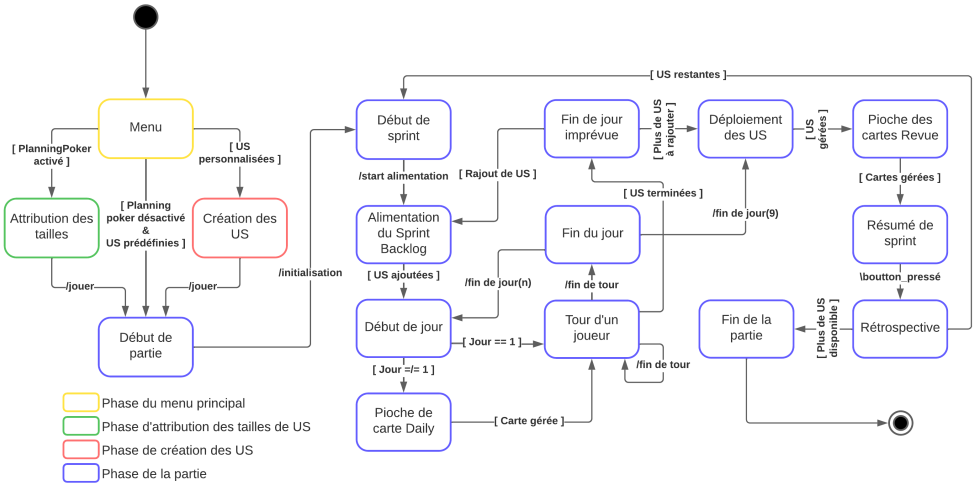


FIGURE 4.1 – Diagramme d'état du logiciel Scrumble

parties, il apparaît tout de même qu'une interaction entre elles est nécessaire. En effet, par exemple, la partie a besoin des tailles choisies par l'équipe lors de la phase d'attribution des tailles, et cette dernière a besoin de connaître la catégorie de US choisie dans le menu principal. Or, toutes ces phases appartenant à des scènes différentes, il n'est pas possible de les faire interagir directement. C'est dans cette problématique qu'apparaît `StateManager`.

4.1.2 Interactions entre les différentes phases

`StateManager`, ou gestionnaire d'état, est une classe statique qui restera active lors de l'exécution des différentes scènes. Son rôle est de stocker les informations relatives à la création de la partie (difficulté, catégorie de US choisie, nombre de joueurs), les tailles de US attribuées lors du Planning Poker, les US créées dans la phase associée ou encore les états de la partie en cours. C'est à cette classe que

chaque phase va s'adresser pour sauvegarder des données ou retrouver des informations nécessaires.
Les différents appels à `StateManager` réalisés par une phase durant son fonctionnement seront disponibles dans les Annexes.

4.1.3 Composants majeurs

Au-delà des différentes phases, il est important d'également définir les objets utilisés par ces dernières tout au long du fonctionnement du logiciel. Ces classes représentent les différents composants physiques que l'on peut retrouver dans une partie réelle de Scrumble.

Carte

La classe `Carte` est la représentation fictive d'une carte du jeu de base. Chaque carte possède une description, une catégorie ainsi qu'un effet particulier. Les cartes sont initialement définies dans un JSON présent dans les fichiers du jeu correspondant aux cartes prévues par Scrumble. Elles sont ensuite instantiées par `CardHandler` une fois la partie lancée en fonction de la langue choisie. Les cartes seront utilisées à de nombreuses reprises lors des différentes phases de pioche de carte. Plus qu'avec une catégorie correspondant à l'intrinsèque de la carte ('Daily', 'Problème' ou 'Revue'), chaque carte possède également un type en fonction de son comportement dans la partie :

1. Simple : Carte devant effectuer une seule action atomique.
2. Multiple : Carte devant effectuer deux actions atomiques.
3. Proposition : Carte devant proposer une série d'actions atomiques et permettant au piocheur d'accepter la proposition ou non.
4. Question : Carte posant une question. Chaque question possède ses propres réponses et son action atomique en cas de bonne ou de mauvaise réponse.
5. Choix : Carte proposant deux séries d'actions atomiques et permettant au piocher de choisir quelle série effectuer.
6. Information : Carte apportant une information et n'ayant pas d'impact sur la partie.

7. Permanente : Carte à durée permanente qui appliquera un effet spécial au fonctionnement normal de la partie. Chaque carte permanente est gérée individuellement tant leurs effets sont particuliers.

Les actions atomiques citées plus haut sont des actions simples telles qu'augmenter ou diminuer les tâches ou la dette technique, passer son tour ou encore piocher des cartes.
Vous trouverez une explication plus complète de cette classe ainsi qu'une liste complète des actions simples aux Annexes B.1 et B.3.

User story

Une représentation des US est également nécessaire. Cette classe va donc permettre de représenter les différentes US qui seront choisies dans une partie. Ces US sont également instantiées par le `StateManager` lors de la phase d'initialisation en fonction de la langue choisie depuis un fichier JSON correspondant. Chaque US possède une description, une taille définie dans les phases précédentes, ainsi qu'une valeur qui lui est propre. Il est important de comptabiliser le nombre de tâches qui ont été attribuées à chaque US et, pour ce faire, une fois rajoutée au Sprint Backlog, chacune d'entre elles recevra un nombre de tâches courant et un nombre de tâches maximum à atteindre, ce dernier variant en fonction de plusieurs paramètres courants à la partie. Chaque US possède également un état représentant son avancée :

- Product Backlog : État correspondant à une US instantiée, mais pas encore ajoutée à un sprint.
- Sprint Backlog : État correspondant à une US ajoutée à un sprint, mais pas encore entamée.
- In Progress : État correspondant à une US ajoutée à un sprint et entamée, mais pas encore terminée.
- Done : État correspondant à une US ajoutée à un sprint et terminée mais pas encore déployée.
- Deployed : État final correspondant à une US terminée et déployée lors de la phase de revue.

Ces états permettent de trier la totalité des US en fonction de leur complétude,

ce qui sera utile plus tard pour proposer un Scrum Board. Vous trouverez une description plus complète des US dans les Annexes.

Dé

Le dé est une simple représentation de la fonction d'un dé. Il est nécessaire à de multiples moments clés du jeu de base ainsi qu'à la gestion des effets de certaines cartes. Le dé est composé d'une liste de sprites représentant ses faces et, lorsqu'il est lancé, il choisit aléatoirement une face parmi cette même liste.

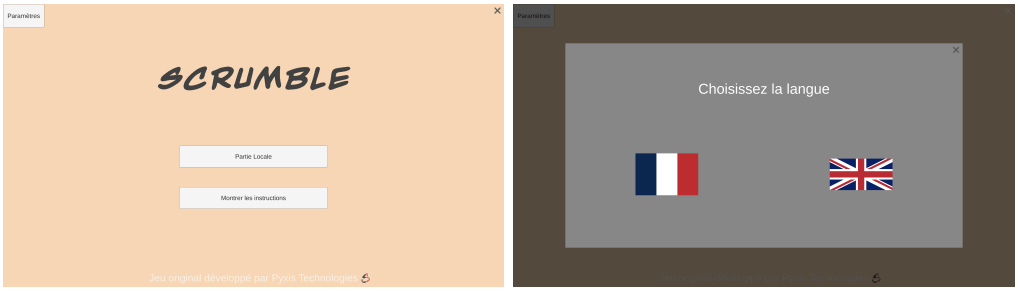
4.2 Menu principal

Le menu principal est la première scène du jeu, la scène d'ouverture. C'est dans cette phase que les joueurs créent la partie en choisissant les options qui les intéressent. Ils y définissent les paramètres que suivra le jeu.
Le menu est lui-même divisé en trois sous-phases :

— L'écran d'accueil
— Les options de la partie
— Le lobby

4.2.1 Écran d'accueil

C'est l'écran dans lequel les joueurs ont accès aux paramètres de l'application et peuvent y changer la langue du jeu (figure 4.2). C'est aussi de là que les joueurs peuvent démarrer une nouvelle partie et accéder aux options de la partie.



(a) Écran principal (b) Écran des paramètres

FIGURE 4.2 – Écrans de l'accueil

US liées

— US 2.5

4.2.2 Options de la partie

C'est l'écran dans lequel les joueurs définissent les paramètres de la partie (figure 4.3).

Ils peuvent y choisir le nom de leur partie, la difficulté, la catégorie des US qui sera utilisée ainsi que s'ils désirent attribuer une taille particulière aux US.

Nom de la partie Utilisé pour personnaliser la partie, il s'agit d'un `string` permettant d'identifier la partie en cours. Il a été rajouté dans une optique d'ajout d'un mode multijoueurs afin de retrouver une partie spécifique de manière plus intuitive.

Difficulté Pouvant être facile, normale ou difficile, la difficulté va intervenir au niveau de la dette technique initiale. Pour une difficulté facile, cette dernière aura une valeur de 5 (et donc un facteur de dette technique de $\times 3$) ; pour une difficulté normale, elle aura une valeur de 15 (et donc un facteur de dette technique de $\times 4$) ; et pour une difficulté difficile, la dette technique vaudra 25 (avec un facteur de $\times 6$). Elle permettra aux joueurs d'adapter la partie en fonction de leur expérience de jeu ou de leur envie de challenge.

Catégorie de user story Plusieurs US sont directement intégrées dans Scrumble et regroupées par catégories, permettant aux utilisateurs de choisir quelles US utiliser sans devoir créer leurs propres US.
Les catégories que peuvent choisir les utilisateurs sont les suivantes :

- Gift Shop (Boutique cadeau) : L'équipe doit fournir une application permettant de gérer une boutique ou d'y acheter des produits.
- Diet Coach (Coach diététique) : L'équipe doit fournir une application permettant de mettre en relation des coachs diététiques et des utilisateurs tout en leur apportant un moyen de suivre la progression de leur régime diététique.
- Travel Diary (Journal de voyage) : L'équipe doit fournir une application permettant aux utilisateurs d'enregistrer leurs voyages, de les partager publiquement ou encore de trouver des voyages et itinéraires en fonction de leurs envies.
- Knowledge Management (Gestion des connaissances) : L'équipe doit fournir une application permettant de gérer un système de gestion de connaissances.
- Custom (Personnalisé) : Ce n'est pas une des catégories de US prédéfinies

que l'équipe utilisera lors de la partie, mais bien une catégorie qu'elle créera elle-même durant la phase de *Création de US*. C'est donc l'option à choisir si l'on veut proposer des user stories personnalisées.

Planing poker Chaque US pré-intégrée possède une taille par défaut. Si les utilisateurs cochent la case de planing poker, ils devront passer par la phase d'*attribution de la taille des US* afin de choisir eux-mêmes la complexité de chacune des US de la catégorie choisie avant de pouvoir commencer la partie.
S'ils ne la cochent pas, les US stories se voient attribuer leur taille par défaut.
Il est à noter que si la catégorie de US choisie est la catégorie *Custom*, la case de planing poker n'a pas d'impact car l'équipe passera de toute façon par une phase de création des US qui leur demandera la taille de chaque US.

FIGURE 4.3 – Options de la partie

US liées

- US 1.2
- US 2.1
- US 2.4

4.2.3 Lobby

C'est l'écran dans lequel les joueurs peuvent voir un récapitulatif des options choisies pour la partie, mais c'est aussi là où ils peuvent ajouter des joueurs au jeu (figure 4.4). Les joueurs ajoutés peuvent modifier leur surnom afin de personnaliser la partie au mieux. Il faut un minimum de 3 joueurs et un maximum de 9 joueurs. Cette étape est également la dernière de la scène du menu principal. Une fois la validation effectuée, le lobby va exporter les options vers le `StateManager` et lancer, en fonction des choix réalisés, la scène suivante adéquate.

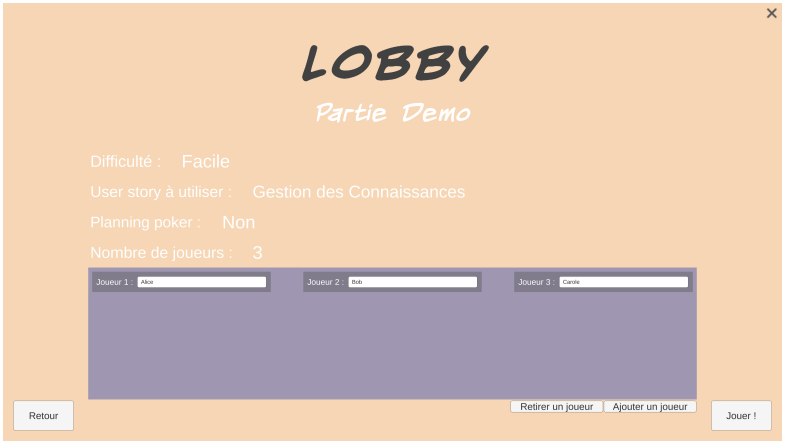


FIGURE 4.4 – Écran du lobby

US liées

— US 1.1

4.2.4 Appels à `StateManager`

Vous trouverez les différents appels au `StateManager` réalisés à la table B.1 des Annexes.

4.3 Attribution de la taille des user stories

Lorsque l'option *Planning poker* a été choisie dans le menu principal, la scène qui suit est l'attribution de la taille des user stories. Comme son nom l'indique, c'est dans cette phase que les joueurs vont décider ensemble de la complexité des US issues d'une même catégorie (figure 4.5). Les US ne prendront dès lors pas leur valeur par défaut et il faudra choisir la taille de chacune d'entre elles avant de pouvoir poursuivre sur la scène suivante : le jeu.
Pour attribuer une taille à une US, les joueurs doivent cliquer sur la US dont ils désirent valider la complexité et ensuite cliquer sur la taille correspondante. Deux états sont donc à représenter : l'état `GLOBAL`, qui représente l'état général où toutes les US sont visibles ; et l'état `PRECISE` qui représente l'état où une US est mise en avant pour lui choisir une taille. Les US dont la taille n'a pas encore été attribuée sont encadrées de rouge alors que celles qui sont terminées sont encadrées de vert, afin de voir rapidement quelles sont les US qui nécessitent encore une réflexion de la part de l'équipe.
Une fois chaque US en possession de sa nouvelle taille, les joueurs peuvent valider leurs choix et commencer la partie.

FIGURE 4.5 – Écrans de l'attribution des tailles de user story

US liées

— US 1.5

4.3.1 Appels à `StateManager`

Vous trouverez les différents appels au `StateManager` réalisés à la table B.2 des Annexes.

4.4 Création des user stories

Lorsque la catégorie de US choisie est `Custom`, le menu principal lance comme deuxième phase la *Création de user stories*. Le but de cette phase est de permettre aux utilisateurs qui le désirent, de créer des US personnalisées. De cette façon, les parties peuvent être plus attractives pour les joueurs, mais aussi plus complètes sans être limitées aux catégories pré-intégrées. Cela permet donc de créer des parties différentes limitées à l'imagination du Product Owner et des participants. Cette phase ressemble fort à la phase d'attribution de taille des US car elle se présente de la même manière (figure 4.6). Toutes les US vierges se retrouvent à l'écran et, en cliquant sur une US qu'on désire remplir, elle est mise en avant afin qu'on puisse y ajouter les caractéristiques désirées. Il y a donc là aussi une distinction particulière à faire entre le moment où les US sont toutes représentées, et le moment où une US est mise en avant pour recevoir des modifications. De manière similaire à la phase d'attribution, la création de US possède donc les états `GLOBAL` et `PRECISE` en fonction des actions des utilisateurs.
Les US vides sont encadrées de rouge, celles incomplètes sont encadrées d'orange alors que les US entièrement complétées sont encadrées de vert. Cela permet de rapidement identifier quelles sont les US qui nécessitent encore une inspection de la part des joueurs.

4.4.1 Caractéristiques à remplir

Voici la liste des caractéristiques à apporter à une user story pour la compléter.

En tant que Cette partie correspond à la personne à qui s'adresse la user story. Elle nécessite un string qui peut être complété par un type d'utilisateur ou une profession.

Description Cela correspond à la description de la US, l'explication de la fonctionnalité recherchée par l'utilisateur défini par la partie *En tant que*. Cette partie est donc complétée par un string.

Valeur La valeur de la US. Elle est comprise entre '1' et '5' et représente l'importance de la fonctionnalité pour le Product Owner mais aussi le nombre de points que les utilisateurs gagneront après avoir achevé cette US. Il y a un total de 45 points à distribuer sur l'ensemble des US créées afin de garder une certaine diversité dans les US.

Taille Elle représente la complexité estimée par l'équipe de développement. De la même manière que la taille peut être choisie dans la phase d'attribution, elle doit être choisie par l'équipe afin de compléter l'US qui y est associée. Elle est donc choisie parmi les valeurs suivantes : XS, S, M, L et XL.

[Restriction] Cette partie est optionnelle. Elle indique si la US en question nécessite qu'une autre US soit déployée au préalable avant de pouvoir être ajoutée à un Sprint Backlog. On y ajoute juste le numéro identifiant de la US pour laquelle on veut rajouter une restriction.

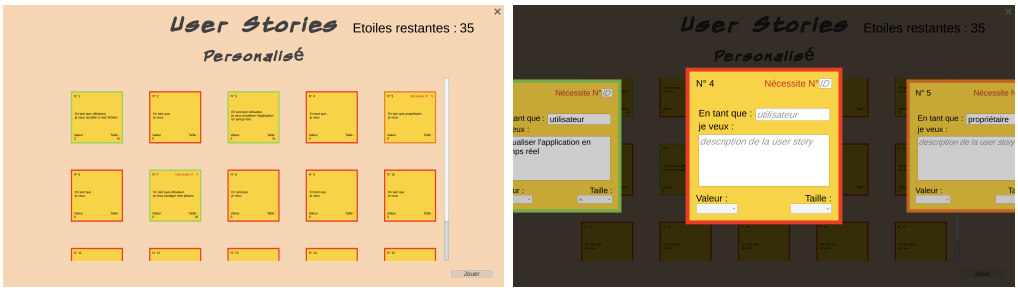


FIGURE 4.6 – Écrans de création des user stories

US liées

— US 1.3
— US 1.4

4.4.2 Appels à `StateManager`

Vous trouverez les différents appels au `StateManager` réalisés à la table B.3 des Annexes.

4.5 Partie

Nous arrivons donc au cœur de la partie en elle-même et c'est là que tout le reste du jeu se déroulera. Ayant choisi une architecture inspirée des microservices, plusieurs intervenants vont alors apparaître afin de gérer au mieux cette dernière scène et les fonctionnalités qui y sont associées. Les intervenants et leurs échanges ainsi que les US auxquelles ils sont liés seront définis à tour de rôle.
Il est dès lors important de définir un peu plus en détails un composant de `StateManager`, `TurnState`. `StateManager` gère l'état global de la partie, mais il gère aussi les variations d'état liées au tour des joueurs. Un diagramme d'état des différentes phases par lesquelles passe le jeu durant le tour d'un joueur est présenté à la figure 4.7.

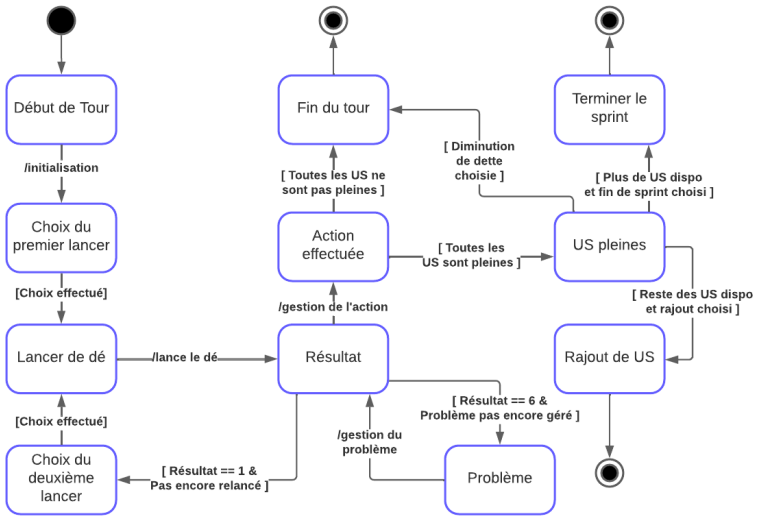


FIGURE 4.7 – Diagramme d'état du tour d'un joueur

4.5.1 `GameManager`

C'est le chef d'orchestre de la phase de jeu. C'est lui qui va gérer les appels aux différents objets en fonction de l'état de la partie. Son rôle est donc plutôt important car il se positionne en tant que lien général entre les différents intervenants et permet la plupart des transitions d'état liées à la scène de la partie représentées à la figure 4.1.
C'est également lui qui se trouve chargé de faire avancer la partie, plus que de simplement gérer les appels. C'est lui qui va gérer la gestion des différents sprints et des différents jours ainsi que du déroulement global du tour d'un joueur. Vous trouverez une image du plan de travail de base à la figure 4.8 ainsi que d'autres captures liées au `GameManager` dans les Annexes.

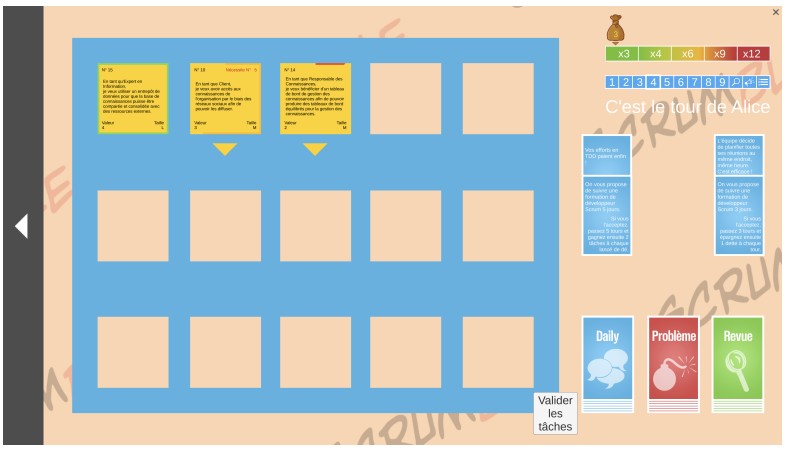


FIGURE 4.8 – Écran de base de la scène de Partie

US liées

- US 1.7
- US 1.8
- US 1.9
- US 1.10
- US 1.11
- US 1.12

— US 1.13
— US 1.14
— US 1.15
— US 1.16
— US 1.17
— US 1.22

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.4 des Annexes.

4.5.2 `SideMenuManager`

Certaines informations doivent être disponibles tout au long de la partie et à chaque instant. De cette nécessité est apparu `SideMenuManager` qui permet dès lors d'afficher un menu alternatif (figure 4.9) regroupant les informations suivantes :

— Le nombre de sprints effectués.
— La durée totale de la partie.
— Le nombre d'étoiles accumulées jusqu'ici.
— Un Scrum Board afin de visualiser l'évolution des différentes US.
— Un bouton permettant d'exporter le Burndown Chart.

Ce menu alternatif est "optionnel" dans le sens qu'il est accessible depuis bon nombre de phases mais qu'il n'est jamais nécessaire de passer par cet écran.

US liées

— US 2.2
— US 2.3

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.5 des Annexes.

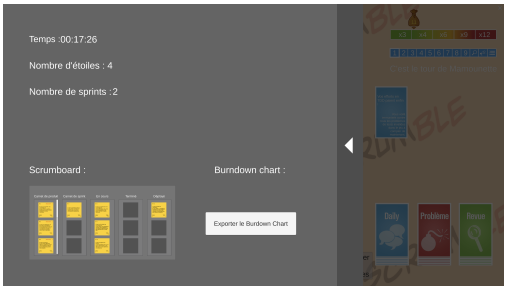


FIGURE 4.9 – Écran du SideMenu

FIGURE 4.10 – Écran du TDDD

4.5.3 `TDDDManager`

`TDDDManager`, ou *ToDo-Doing-Done Manager*, est la partie du jeu où les membres de l'équipe de développement peuvent alimenter le Sprint Backlog en choisissant des US sur lesquelles travailler, issues du Product Backlog (figure 4.10). Les US y sont séparées en trois catégories : À faire (ToDo), En cours (Doing) et Terminé (Done) ; et il n'est possible de déplacer que les US ayant l'état Product Backlog et dont la US de restriction, si elles en possèdent une, a déjà été déployée. Cette phase intervient à deux moments dans la partie : quand toutes les US du sprint ont été terminées et que l'équipe décide de réalimenter le Sprint Backlog, et quand un nouveau sprint débute.

US liées

— US 1.6
— US 1.15

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.6 des Annexes.

4.5.4 `CardHandler` et `CardPicker`

`CardHandler` et `CardPicker` sont chargés de gérer les pioches de cartes quand cela est nécessaire. `CardPicker` récupère des cartes depuis les pioches et gère donc

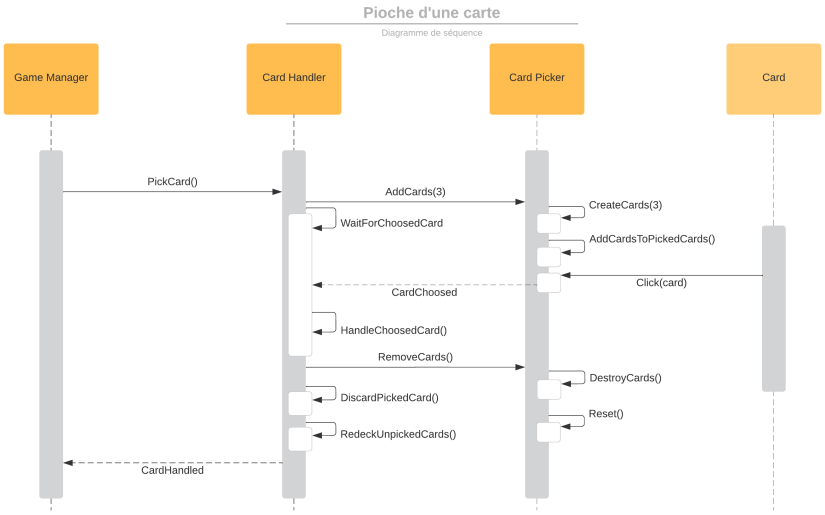


FIGURE 4.11 – Diagramme de séquence de la pioche d'une carte

les cartes présentes physiquement, alors que `CardHandler` s'occupe de gérer les effets des cartes une fois qu'elles sont piochées. Leurs rôles sont donc bien différents mais très complémentaires. Vous trouverez une représentation des appels entre `GameManager`, `CardHandler` et `CardPicker` lors de la nécessité de piocher une carte à la figure 4.11 ainsi que quelques images illustrant les différents écrans rencontrés lors de cette phase à la figure 4.12.

US liées

— US 1.13
— US 1.14
— US 1.19

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.7 des Annexes.

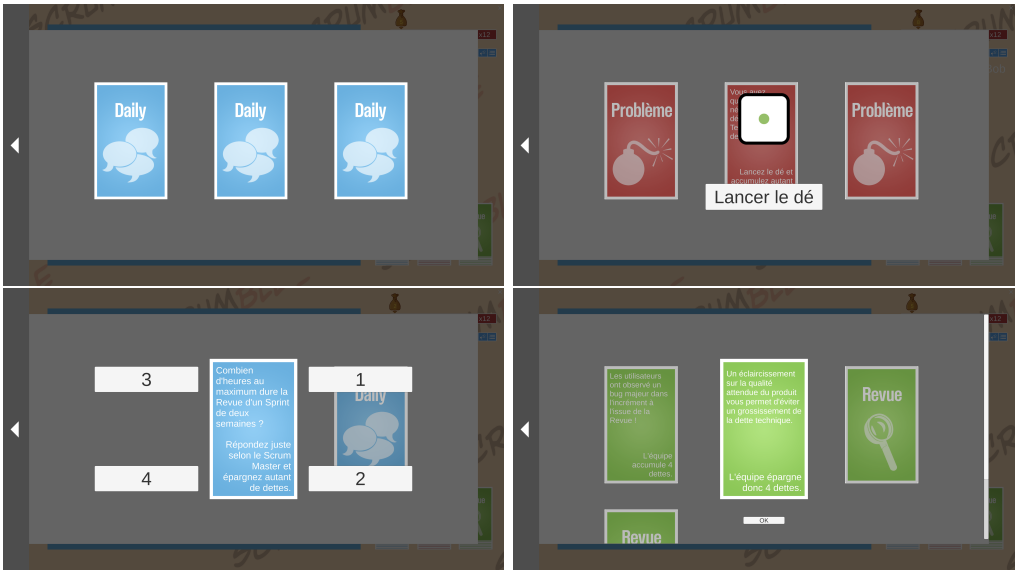


FIGURE 4.12 – Écrans liés à la pioche des cartes et à leur gestion

4.5.5 `ReviewManager`

`ReviewManager` s'occupe de gérer toute la phase de revue. Dans un premier temps, il récupère l'ensemble des US présentes dans l'espace de travail du `GameManager` afin de les évaluer. Si une US est terminée, c'est-à-dire que son nombre courant de tâches est égal à son nombre de tâches maximum, la US passe de l'état `Done` à l'état `Deployed`. Si une US n'est pas terminée, le `ReviewManager` replace la US dans l'espace de travail et augmente la dette technique d'un point par tranche de cinq tâches non-réalisées (figure 4.13). Une fois chaque US analysée, les joueurs doivent piocher autant de cartes 'Revue' que de US déployées. Ces cartes seront gérées par le `CardHandler` et le `CardPicker`. Une fois tous les effets des cartes appliqués, `ReviewManager` cède sa place à `SummaryManager`.

US liées

- US 1.18
- US 1.19
- US 1.20

FIGURE 4.13 – Gestion des user stories lors de la phase de revue

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.8 des Annexes.

4.5.6 `SummaryManager`

Afin de satisfaire la US 2.7, il était nécessaire de rajouter un écran spécial reprenant les différentes informations relatives au sprint achevé. `SummaryManager` récupère ces informations depuis le `StateManager` et affiche un résumé du sprint (figure 4.14). Voici une description des informations que l'on peut obtenir dans cet écran :

- Durée du sprint : Le temps écoulé entre le début du sprint et l'affichage du résumé.
- Valeur accumulée : Le nombre d'étoiles qui étaient récupérées au début du sprint et le nombre d'étoiles récupérées à la fin du sprint.
- Dette technique courante : La valeur de la dette technique au début du sprint et la valeur de la dette technique à la fin du sprint.
- Nombre de US terminées : Le nombre de US terminées au début du sprint et le nombre de US terminées à la fin du sprint.
- Nombre de tâches qui ont été perdues lors du sprint.
- Nombre de cartes "Problème" qui ont été piochées.

À cela se rajoutent un bouton permettant d'exporter le Burndown Chart actualisé ainsi qu'un "Scrumboard" (plus d'informations dans les sections dédiées). La phase de revue se termine effectivement à la fin de cette phase de résumé. La phase qui

suit la revue est la rétrospective.

US liées

— US 2.2
— US 2.3
— US 2.7

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.9 des Annexes.

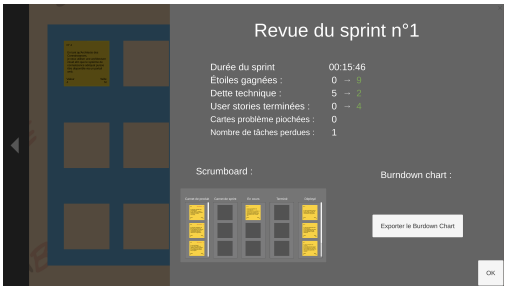


FIGURE 4.14 – Écran de résumé de sprint

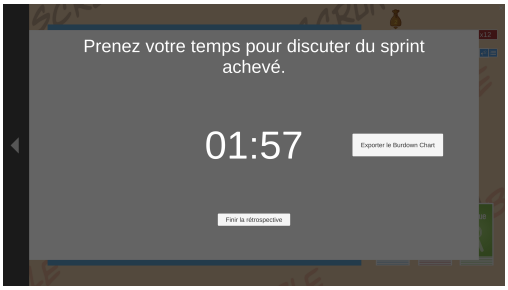


FIGURE 4.15 – Écran de la phase de rétrospective

4.5.7 `RetrospectiveManager`

Lorsque la partie atteint l'état de rétrospective, il est nécessaire de fournir du temps au Scrum Master afin qu'il puisse donner des retours sur le déroulement de la partie à l'équipe. C'est le rôle de `RetrospectiveManager` que de proposer ce temps, par défaut de deux minutes, mais aussi de permettre d'écourter ce temps si ce dernier est trop long. Est également disponible dans cette phase, un bouton permettant d'exporter le Burndown Chart afin de pouvoir s'appuyer sur des valeurs réelles et actualisées (figure 4.15).

US liées

— US 1.21

— US 2.2

Appels à `StateManager`

La phase de rétrospective ne fait pas d'appel direct à `StateManager`.

4.5.8 `EndScreenManager`

Une fois la partie terminée, `EndScreenManager` se charge d'afficher un récapitulatif du jeu en y affichant les options choisies au menu principal ou encore les joueurs, mais en y affichant aussi des informations propres au déroulement de la partie telles que le nombre de sprints effectués ou la durée totale de la partie (figure 4.16). C'est aussi lui qui se charge d'exporter la preuve de complétion en prenant une capture d'écran de l'écran de fin du jeu et en permettant à l'utilisateur de la sauvegarder dans son ordinateur.

US liées

FIGURE 4.16 – Écran de fin de partie

— US 2.2
— US 2.6

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.10 des Annexes.

4.5.9 `BurndownChartManager`

Le rôle de `BurndownChartManager` est de générer un fichier CSV contenant les différentes attributions de tâches aux US au fur et à mesure de la partie. Il permet de suivre l'évolution actualisée du jeu de manière quantitative tout en proposant une complétion théorique et "idéale" de l'ensemble des US présentes au début d'un sprint. Afin de permettre l'export de ce document, il a été nécessaire de rajouter une bibliothèque de gestion de fichier permettant de choisir le chemin d'export en fonction de l'environnement utilisé : Standalone File Browser[17]. `BurndownChartManager` s'actualise également dans le cas où le nombre de tâches maximum des US est diminué par une carte permanente ou si des tâches sont à rajouter au début d'un nouveau sprint.
Des boutons permettant d'exporter le Burndown Chart se trouvent dans le menu alternatif, dans la phase de résumé, dans la phase de rétrospective ainsi que dans la phase finale, la fin de partie.

US liées

— US 2.2

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.11 des Annexes.

4.5.10 `ScrumboardManager`

L'élaboration d'un Scrum Board permet à l'équipe de visualiser au mieux l'évolution des US à faire, en cours et terminées. C'est pourquoi `ScrumboardManager` s'occupe de générer ce tableau en fonction de l'état de chaque US présente dans la

partie en les divisant en cinq catégories correspondant aux états que peut avoir une user story : `Product Backlog`, `Sprint Backlog`, `In Progress`, `Done` et `Deployed` (figure 4.17). Il est impossible d'y déplacer les US ou de modifier leur état, ce tableau est seulement disponible à titre informatif dans le menu alternatif et lors de la phase résumant le sprint achevé.

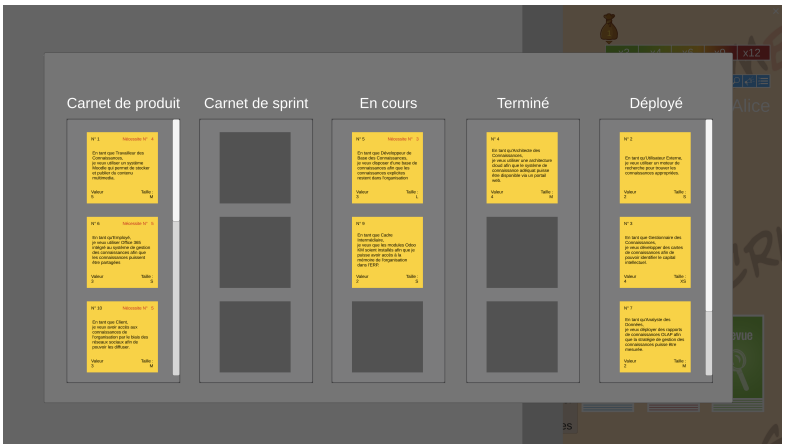


FIGURE 4.17 – Écran affichant le Scrum Board

US liées

— US 2.3

Appels à `StateManager`

Les différents appels au `StateManager` réalisés se trouvent à la table B.12 des Annexes.

4.5.11 `AnimationManager`

Toutes ces phases font apparaître et disparaître beaucoup d'éléments visuels. Afin de rendre ces déplacements plus fluides et agréables, il était nécessaire d'attribuer une fonction de gestion des animations à un service dédié. `AnimationManager` s'occupe donc de gérer toutes les animations de la partie. Chaque élément nécessitant une animation possède un lien direct vers ce service et lui transmet toutes les informations nécessaires à l'animation désirée. Afin de réaliser des animations

encore plus fluides et moins sacadées, il a été décidé de rajouter une bibliothèque de fonctions d'accélération, ou *easing functions*[23]. Ces fonctions permettent des mouvements avec des vitesses changeantes en fonction du temps, ce qui les rend plus naturels.

Appels à `StateManager`

Aucun appel direct à `StateManager` n'est effectué depuis `AnimationManager`.

4.5.12 `ExitManager`

Le dernier composant que nous verrons ici est `ExitManager`. Quand un joueur clique sur la croix au coin supérieur droit de l'écran, `ExitManager` affiche une fenêtre demandant confirmation avant de terminer la partie (figure 4.18). Cette croix de fermeture du logiciel est accessible à tout moment du jeu.

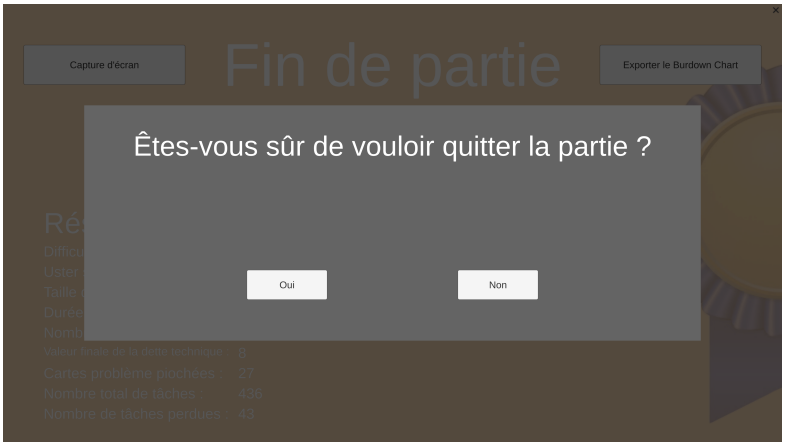


FIGURE 4.18 – Écran de fermeture de Scrumble

Appels à `StateManager`

`ExitManager` n'effectue aucun appel à `StateManager`.

Chapitre 5

Tests

5.1 Tests de méthode Scrum

Dans le déroulement standard d'une méthode Scrum, et de manière plus générale dans un cycle itératif, les tests prennent une place particulière. Lorsque l'on choisit des US à implémenter lors d'un sprint, il est important de s'assurer que les fonctionnalités développées soient opérationnelles avant de rajouter de nouvelles US au Sprint Backlog. Il convient donc de vérifier par de nombreux tests le bon fonctionnement de chaque US terminée afin de ne pas devoir revenir sur des US déjà achevées et déployées.
Durant les différentes phases de développement, chaque US a donc été testée de manière complète afin de s'assurer que son comportement se rapproche au mieux de la fonctionnalité désirée. Ces tests ont été réalisés en analysant toutes les différentes situations auxquelles le service rajouté pourrait faire face et en analysant son comportement permettant une intégration continue.

5.2 Tests d'export en autonome

Une fois toutes les fonctionnalités ajoutées et testées, il a fallu vérifier le bon comportement du logiciel de manière autonome et sur les différents environnements proposés. Cela a permis de mettre en évidence certains soucis relatifs aux plateformes utilisées, notamment la différence de qualité d'écran et le nombre de pixels

qui en découlent. Il a donc été nécessaire de retravailler certaines fonctionnalités relatives aux animations ou aux accès à l'ordinateur en lui-même.

5.3 Tests à plus grande échelle

Une fois tous les tests individuels réalisés, il était important de faire tester l'application par un grand nombre d'utilisateurs afin de recevoir des retours quant à son fonctionnement. Certains bugs mineurs ont ainsi été corrigés et ont permis d'améliorer le fonctionnement du jeu, alors que d'autres retours ont permis de modifier son côté graphique et visuel.
Une fois cette dernière série de tests achevée, le logiciel est alors prêt à être déployé et utilisé.

Chapitre 6

Pistes d'amélioration

Plusieurs pistes d'amélioration restent tout de même envisageables afin de rendre l'application encore plus complète. Elles vous seront décrites un peu plus en détail dans ce chapitre.

6.1 Multijoueurs

La première piste d'amélioration pourrait être de rendre l'application utilisable par plusieurs joueurs sur plusieurs appareils en même temps. Au lieu de se connecter sur une machine unique qui fait tourner Scrumble, les utilisateurs pourraient se connecter et jouer ensemble sur une partie commune où chaque personne pourrait jouer à son propre tour. Cela est tout à fait possible et plusieurs manières de procéder ont même été explorées.
En hébergeant une partie sur sa machine, un utilisateur "hébergeur" pourrait inviter d'autres joueurs à rejoindre sa partie, évitant ainsi la nécessité d'un serveur externe.
On pourrait également héberger la partie sur un serveur afin que tous les joueurs puissent s'y connecter sans créer un hébergement. Unity propose une série de serveurs personnels de manière gratuite tant que l'application n'est pas trop gourmande en terme d'échange de ressources et en terme de nombre d'utilisateurs.

6.2 Interface graphique

L'interface graphique courante de l'application pourrait bien évidemment être retravaillée. Changer les couleurs utilisées, les formes des éléments présents ou encore changer la police d'écriture, toutes ces améliorations graphiques sont bien évidemment prêtes à évoluer afin de rendre le jeu plus attractif et encourager le déroulement du jeu. Rajouter une bande son et des éléments sonores en fonction des différentes actions effectuées par les utilisateurs pourrait également rendre l'application plus conviviale.

6.3 Burndown Chart graphique

Le Burndown Chart est actuellement exporté dans un fichier CSV, aucun graphe n'y est donc affiché. Une amélioration pourrait être d'intégrer un graphe dans ce fichier CSV, ou même de l'intégrer dans Scrumble directement afin de visualiser plus efficacement les données qui y sont liées. Cela pourrait être réalisé par des bibliothèques payantes disponibles dans Unity ou en développant soi-même sa propre librairie de création de graphe.

6.4 Personnalisation à la taille de l'écran

Actuellement, le jeu est basé sur une résolution de 1920x1080 pixels et donc un rapport d'écran de 16/9. Cette dimension est respectée dans tous les environnements, mais si l'application est lancée depuis un écran avec des dimensions natives différentes, les éléments graphiques ne s'affichent plus aux bons endroits. Il est donc déconseillé pour le moment de jouer avec un rapport longueur/largeur différent de 16/9. Une amélioration pourrait donc être de permettre aux utilisateurs de choisir le rapport et la qualité d'écran qui leur correspond.

6.5 Importer des UserStories

Des user stories peuvent être créées directement depuis le jeu par la phase de "Création des User Stories", mais cette phase peut être plutôt longue pour les

utilisateurs et empêcher le partage rapide de nouvelles user stories. Il pourrait être intéressant de rajouter un moyen d'importer une série de user stories depuis un fichier externe afin de rajouter rapidement de nouvelles catégories de US.

6.6 Boutons d'aide

Une dernière amélioration pourrait être de rajouter des boutons d'aide pour l'utilisateur. Le manuel d'utilisation est fourni dans le jeu, mais des petites indications de rappel pourraient être les bienvenues pour un joueur perdu. Cela permettrait de rendre le jeu plus accessible et dynamique pour les nouveaux joueurs.

Chapitre 7

Conclusion

L'objectif premier était de proposer un logiciel permettant à ses utilisateurs de réaliser une partie du Serious Game "Scrumble". Il vous a été montré au travers de ce travail de fin d'études les différentes étapes liées à l'élaboration dudit logiciel, mais aussi les enjeux auxquels il fait face.

Enjeux et place du logiciel Au-delà de la livraison d'un logiciel fonctionnel, ce mémoire se positionne dans une optique de proposition d'un nouveau moyen de partager de l'information dans un monde qui évolue et qui nécessite des méthodes d'apprentissage en phase avec les exigences contemporaines. Il a été montré que les jeux sérieux peuvent bien répondre à la problématique et proposer une manière novatrice de faire comprendre des concepts majeurs à leurs utilisateurs.

Scrum pour un jeu Scrum Lors des choix d'implémentation du logiciel, il a été décidé d'organiser le travail sous la méthode de développement Scrum. Au-delà de la mise en abîme, il était intéressant de voir l'efficacité de cette méthode dans la conception d'un projet comme celui-là. Les limites de Scrum et, de manière plus globale, des méthodes de développement basées sur de multiples itérations, sont vite apparues.
Premièrement, ces méthodes fonctionnent mieux à plusieurs. Scrum nécessite un minimum de trois rôles pour fonctionner : le Product Owner, ici joué par mon promoteur Monsieur Kolp ; l'équipe de développement, mon rôle en tant que déve-

loppeur du logiciel ; ainsi que le Scrum Master, personne expérimentée et formée à la méthode Scrum et, plus largement, à la méthodologie Agile. Sans Scrum Master, la méthode Scrum ne peut s'exécuter correctement.
Aussi, la première valeur définie par le Manifeste Agile, privilégier les individus et leurs interactions plus que le processus et les outils, perd tout son sens dans le cadre d'un travail en solitaire. Les méthodes agiles favorisent la discussion et la communication car elles apportent beaucoup à l'évolution du projet : feedback pour les prochaines itérations, adaptabilité sur les problèmes rencontrés et sur l'évolution ou encore création d'un environnement de travail sain.
Certaines valeurs des méthodes agiles ont cependant été très utiles, telles que l'intégration continue ou la création de user stories, en rajoutant une structure au développement.

Une approche qui aurait pu être plus judicieuse pour le développement de ce projet, aurait été de travailler sous un cycle de développement basé sur le modèle en cascade amélioré. Il accorde moins d'adaptabilité qu'un modèle itératif, mais il apporte une structure plus judicieuse pour un travail en solitaire tout en permettant de bien définir les exigences liées au système et au logiciel.

État final du logiciel Au terme du développement de ce projet, le produit fini est fonctionnel et répond à toutes les fonctionnalités qui composaient le cahier des charges. Cette première version est bien entendu sujette à des pistes d'améliorations telles que celles évoquées dans le chapitre éponyme, mais elle représente une base fonctionnelle permettant de jouer une partie de Scrumble, objectif premier de ce mémoire.

Bibliographie

[1] A guide to scrum body of knowledge. https://www.scrumstudy.com/Chapter%201%20(Introduction).pdf, 2013.

[2] Comprendre la méthode agile scrum en 10 minutes. https://www.tuleap.org/fr/agile/comprendre-methode-agile-scrum-10-minutes, Apr 2023.

[3] What is scrum. https://scrumguides.org/, 2023.

[4] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods : Review and analysis. *arXiv (Cornell University)*, 1 2002.

[5] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods : Review and analysis, 2017.

[6] David J. Anderson and Andy Carmichael. *Essential Kanban condensed*. Lean-Kanban University, 11 2015.

[7] Renae Aurisch, Mohiuddin Ahmed, and Abu Barkat Ullah. An outlook at agile methodologies for the independent games developer. *International Journal of Computers and Applications*, 43 :1–7, 05 2019.

[8] Kent Beck and Cynthia Andres. *Extreme Programming explained : Embrace Change (2nd Edition)*. 11 2004.

[9] Luiz Fernando Capretz. Y : a new Component-Based software life cycle model. *Journal of Computer Science*, 1(1) :76–82, 1 2005.

[10] Claire Cornic. Qu'est-ce que le planning poker ? https://blog-gestion-de-projet.com/quest-ce-que-le-planning-poker/, Apr 2023.

[11] Ward Cunningham. Manifeste pour le développement agile de logiciels. http://agilemanifesto.org/iso/fr/manifesto.html, 2001.

[12] Darina Dicheva, Christo Dichev, Gennady Agre, and Galia Angelova. Gamification in education : A systematic mapping study. *Educational Technology & Society*, 18 :75–88, 07 2015.

[13] DonWells. Extreme programming.svg. https://commons.wikimedia.org/wiki/File:Extreme_Programming.svg, 7 2013.

[14] Christiane Eichenberg and Markus Schott. Serious games for psychotherapy : A systematic review. *Games for Health Journal*, 6(3) :127–135, 2017. PMID : 28628385.

[15] Alain Fernandez. Qu'est-ce que scrum, méthode de développement agile. https://www.piloter.org/projet/methode/scrum.htm, Apr 2023.

[16] Sarah Gentry, A. Gauthier, Beatrice L'Estrade Ehrstrom, David Wortley, Anneliese Lilienthal, Lorainne Tudor Car, Shoko Dauwels-Okutsu, Charoula Konstantia Nikolaou, Nabil Zary, James Campbell, and Josip Car. Serious Gaming and Gamification Education in Health Professions : Systematic review. *Journal of Medical Internet Research*, 21(3) :e12994, 3 2019.

[17] Gkngkc. GitHub - gkngkc/UnityStandaloneFileBrowser : A native file browser for unity standalone platforms. https://github.com/gkngkc/UnityStandaloneFileBrowser.

[18] David Hinde. *PRINCE2 Study Guide*. 3 2012.

[19] Andy Hunt. The failure of agile. https://toolshed.com/2015/05/the-failure-of-agile.html, Jun 2015.

[20] Marko Ikonen, Elena Pirinen, Fabian Fagerholm, Petri Kettunen, and Pekka Abrahamsson. On the impact of kanban on software project work an empirical case study investigation. pages 305–314, 04 2011.

[21] Ron Jeffries. Developers should abandon agile. https://ronjeffries.com/articles/018-01ff/abandon-1/, May 2018.

[22] Rémi Lardilleux. Cycle en v en gestion de projet : Définition et méthode. https://www.manager-go.com/gestion-de-projet/cycle-en-v.htm, Feb 2021.

[23] Markv12. ManagingCoroutines/Assets/Scripts/Easing.cs at master · markv12/ManagingCoroutines. https://github.com/markv12/ManagingCoroutines/blob/master/Assets/Scripts/Easing.cs.

[24] Shannon L. Marlow, Eduardo Salas, Lauren B. Landon, and Bart Presnell. Eliciting teamwork with game attributes : A systematic review and research agenda. *Computers in Human Behavior*, 55 :413–423, 2016.

[25] Charlotte Moreau. Analyse des intérets et des enjeux d'adopter les méthodes agiles pour innover au sein d'une entreprise, cas pratique chez colruyt group, 2016. Prom. : Latiers, Mélanie. http://hdl.handle.net/2078.1/thesis:3996.

[26] Pyxis. Scrumble - An Agile board game to discover and improve your skills as a Scrum team ! www.scrumble.pyxis-tech.com/index.html.

[27] Kenneth H. Rose. A Guide to the Project Management Body of Knowledge (PMBOK® Guide)-Fifth Edition. *Project Management Journal*, 44(3) :e1, 5 2013.

[28] V.M. Rota and J. Tabaka. *Gestion de projet : vers les méthodes agiles.* Architecte logiciel. Eyrolles, 1 2007.

[29] W. W. Royce. Managing the development of large software systems : Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, page 328–338, Washington, DC, USA, 1987. IEEE Computer Society Press.

[30] Ben Schaefer. Into the scrum. *PM Network*, 29(12) :16–17, Dec 2015.

[31] Ken Schwaber. *SCRUM development process.* 1 1997.

Annexe A

Écrans du jeu

FIGURE A.1 – Écran d'accueil

FIGURE A.2 – Écran des paramètres

FIGURE A.3 – Écran des options

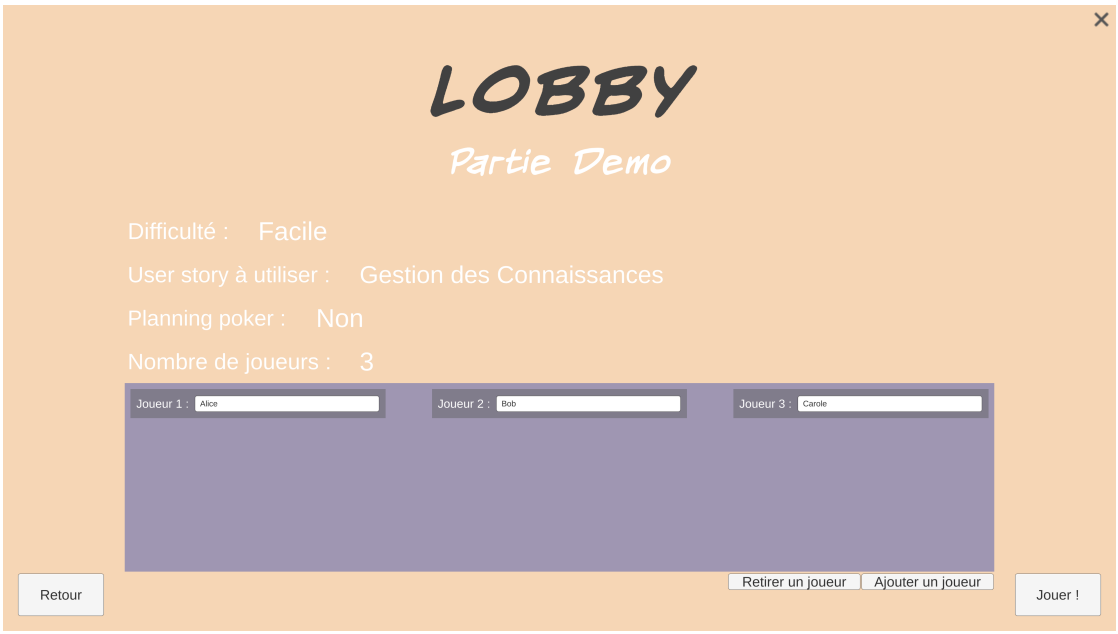


FIGURE A.4 – Écran du lobby

FIGURE A.5 – Écran d'attribution des tailles d'US général

FIGURE A.6 – Écran d'attribution des tailles d'US précis

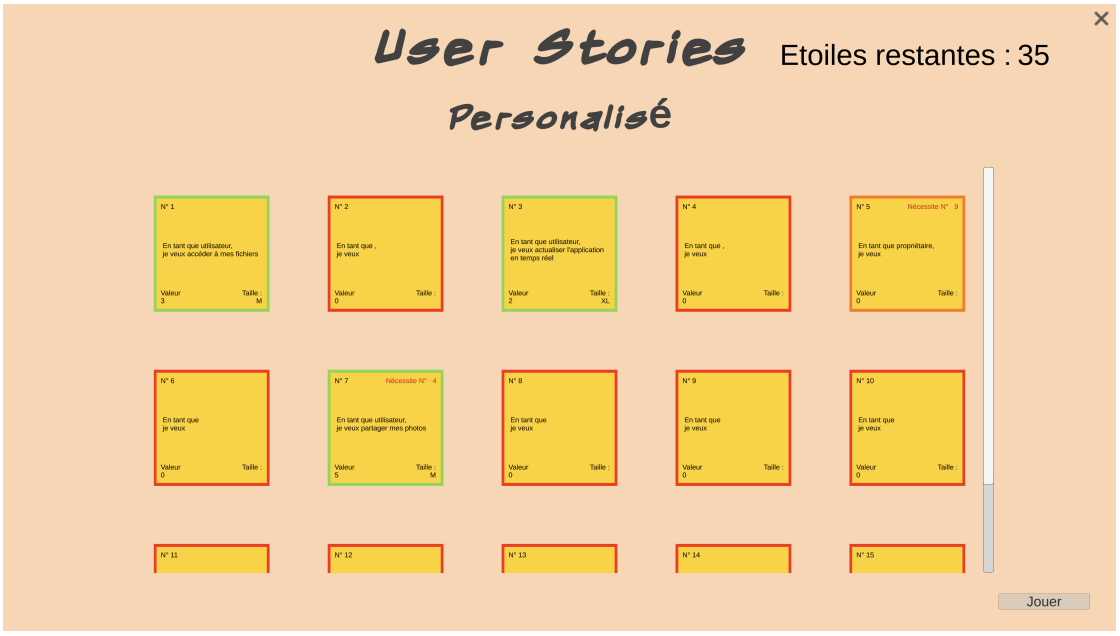


FIGURE A.7 – Écran de création des US général

FIGURE A.8 – Écran de création des US précis

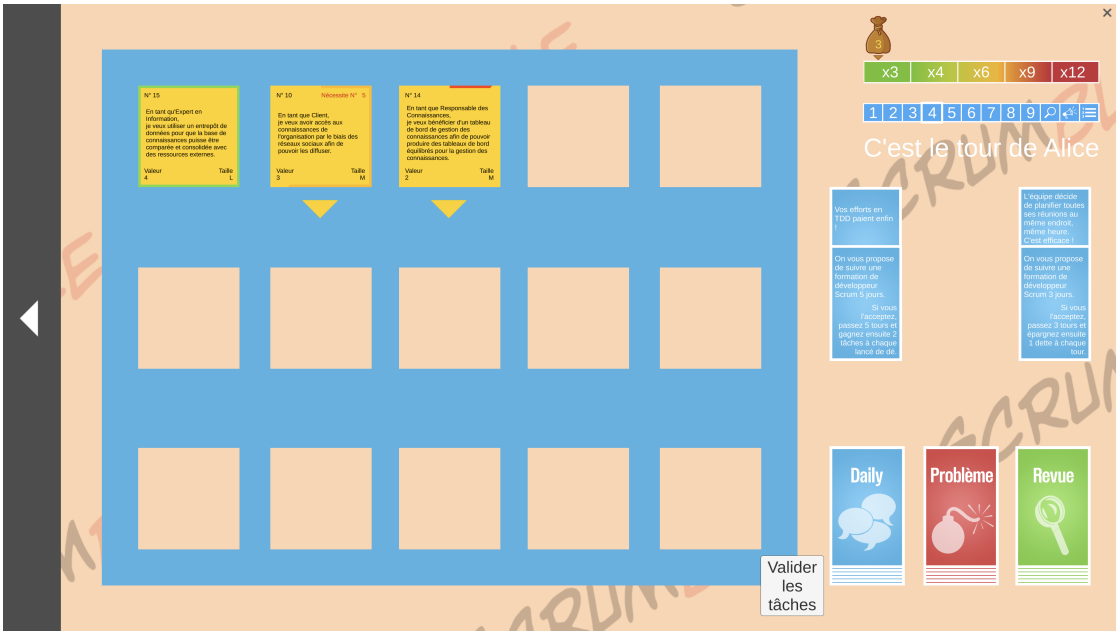


FIGURE A.9 – Écran général de la Partie

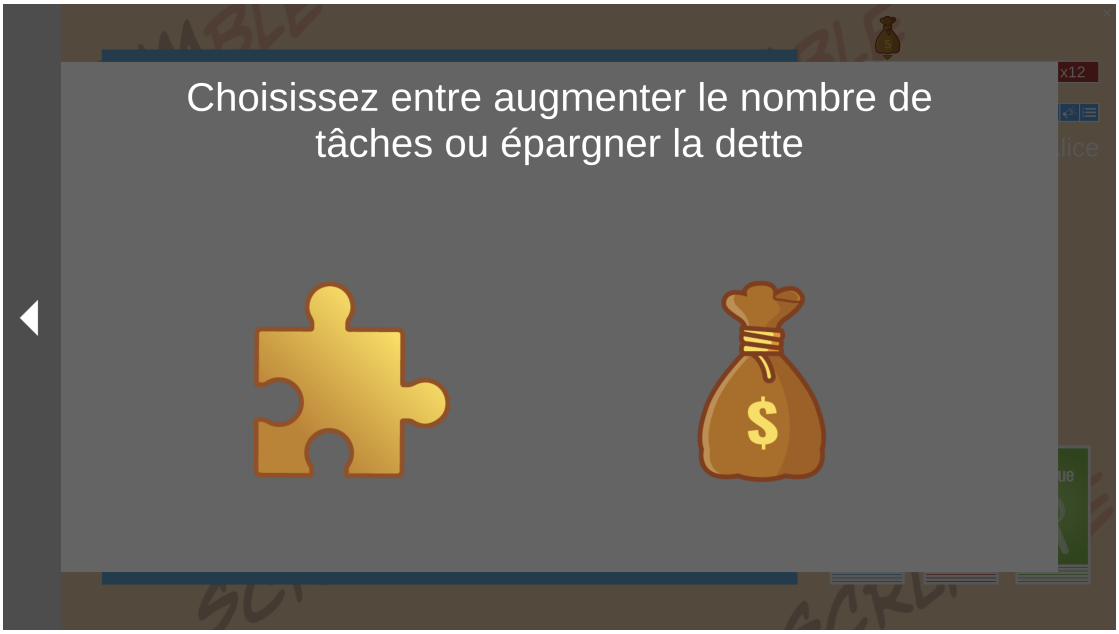


FIGURE A.10 – Écran montrant le choix entre ajouter des tâches ou diminuer la dette technique

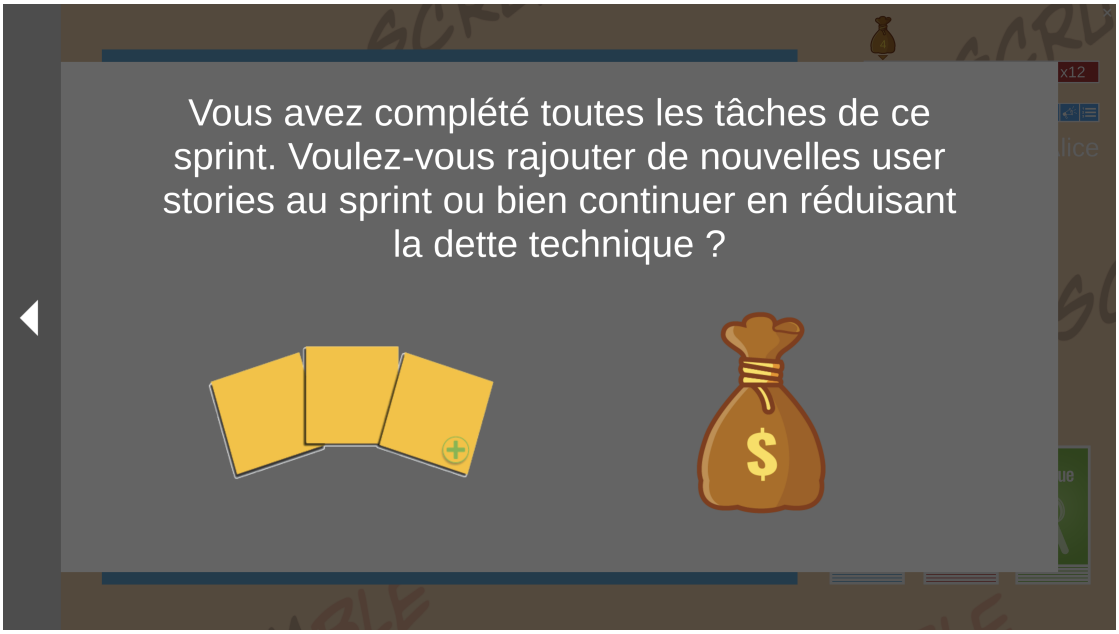


FIGURE A.11 – Écran montrant le choix entre rajouter de nouvelles US au Sprint Backlog ou diminuer la dette technique

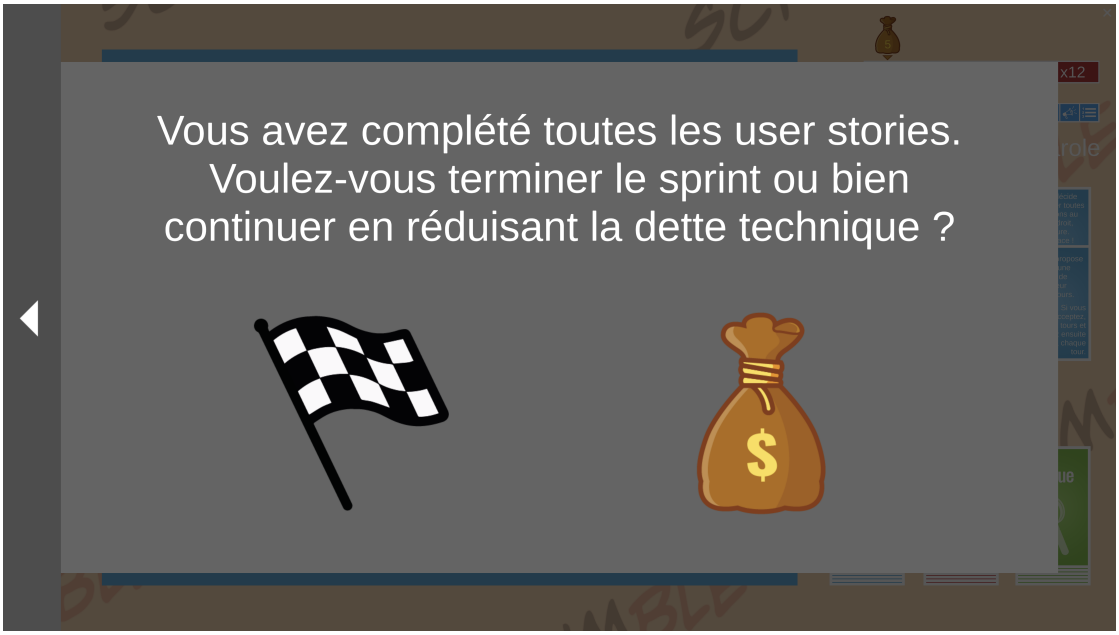


FIGURE A.12 – Écran montrant le choix entre terminer la partie ou diminuer la dette technique

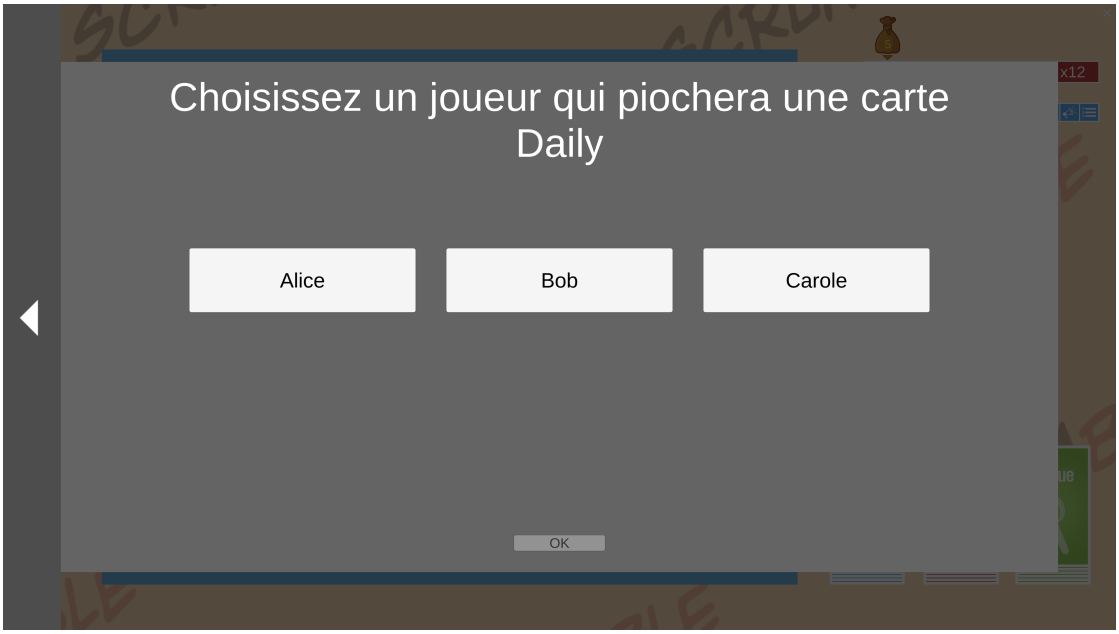


FIGURE A.13 – Écran de sélection du joueur devant piocher la carte Daily

FIGURE A.14 – Écran du lancer de dé

FIGURE A.15 – Écran du ToDo-Doing-Done

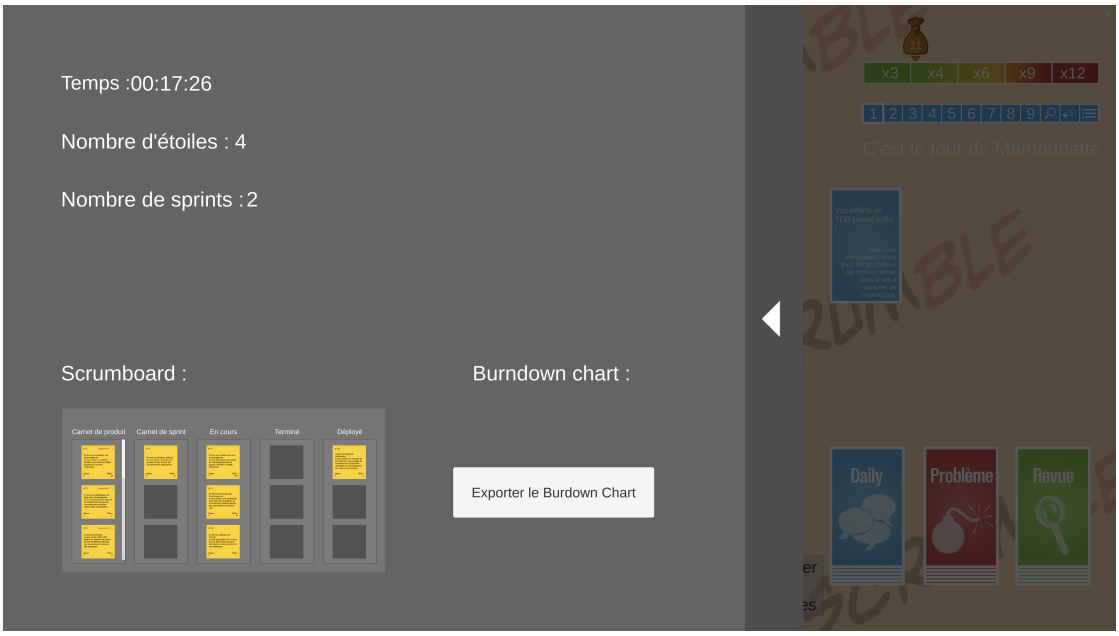


FIGURE A.16 – Écran du menu alternatif

Annexe B

Complément d'architecture

B.1 Card

Au listing B.1 vous trouverez un modèle JSON complet d'une carte pour Scrumble.

- `id` : Le numéro de la carte dans sa catégorie.
- `category` : La catégorie de la carte. Elle peut être `PROBLEM`, `DAILY` ou `REVIEW`.
- `description` : La description de la carte, c'est-à-dire la partie descriptive. Elle donne un contexte à la carte.
- `[result]` : Le résultat de la carte, c'est-à-dire la partie concernant l'effet de la carte et ce qu'il en découlera.
- `typeOfCard` : Le type de la carte en rapport avec son action. Il peut être `Simple`, `Multiple`, `Question`, `Permanent`, `Choice`, `Information` ou `Proposition`.
- `[firstAction]` : La première action atomique de la carte.
- `[firstValue]` : La première valeur associée à la première action de la carte.
- `[secondAction]` : La deuxième action atomique de la carte.
- `[secondValue]` : La deuxième valeur associée à la deuxième action de la carte.
- `[thirdAction]` : La troisième action atomique de la carte.
- `[thirdValue]` : La troisième valeur associée à la troisième action de la

carte.

- `[permanent]` : Le type de carte permanente.
- `[questionId]` : Le numéro de la question que représente la carte.
- `[positive]` : Est-ce que la carte est de catégorie `DAILY` et a un effet positif ?
- `[test]` : Est-ce que la carte est de catégorie `PROBLEM` et est en rapport avec un échec de test ?
- `[answer1]` : La première réponse si la carte est une question.
- `[answer2]` : La deuxième réponse si la carte est une question.
- `[answer3]` : La troisième réponse si la carte est une question.
- `[answer4]` : La quatrième réponse si la carte est une question.

Listing B.1 – Modèle JSON de Card

```
{
    "id" = <int>;
    "category" = <Category>;
    "description" = <string>;
    "result" = <string>;
    "typeOfCard" = <TypeOfCard>;
    "firstAction" = <Action>;
    "firstValue" = <float>;
    "secondAction" = <Action>;
    "secondValue" = <float>;
    "thirdAction" = <Action>;
    "thirdValue" = <float>;
    "permanent" = <Permanent>;
    "questionId" = <int>;
    "positive" = <bool>;
    "test" = <bool>;
    "answer1" = <string>;
    "answer2" = <string>;
    "answer3" = <string>;
    "answer4" = <string>;
}
```

Listing B.2 – Enum de TypeOfCard

```
public enum TypeOfCard{
        None,
        Simple,
        Multiple,
        Question,
        Permanent,
        Choice,
        Information,
        Proposition,
    }
```

Listing B.3 – Enum de Action

```
public enum Action{
        None,
        IncreaseTask,
        DecreaseTask,
        IncreaseDebt,
        DecreaseDebt,
        IncreaseTaskPerPlayer,
        IncreaseDebtPerPlayer,
        DecreaseTaskPerPlayer,
        DecreaseDebtPerPlayer,
        IncreaseTaskPerDevelopper,
        IncreaseDebtPerDevelopper,
        DecreaseTaskPerDevelopper,
        DecreaseDebtPerDevelopper,
        MultiplieDebt,
        DecreaseTaskPerCurrentDebt,
        IncreaseTaskPerRoll,
        DecreaseTaskPerRoll,
        IncreaseDebtPerRoll,
        DecreaseDebtPerRoll,
        CurrentPlayerPassATurn,
        NextPlayerPassATurn,
```

```
23          AllPlayersPassATurn,
24          PickDailyCards,
25          PickProblemCards,
26          PickReviewCards,
27          PickProblemCardsPerRoll,
28          GetRidOfJinxCard,
29          DecreaseMaxTaskNextSprint,
30          IncreaseTaskNextSprint,
31      }
```

Listing B.4 – Enum de Permanent

```
1  public enum Permanent{
2           None,
3           NoMoreTestIssues,
4           OneMoreTaskPerRoll,
5           TasksOnBeginSprint,
6           TwoMoreTasksPerRoll,
7           MaxUserStoriesLowered,
8           DecreaseDebtPerTurn,
9           Jinx,
10          SkipProblemOrDoubleDaily,
11          OneTaskPerDay,
12      }
```

B.2 UserStory

Au listing B.5 vous trouverez une représentation du modèle d'une user story selon Scrumble.

- `id` : Le numéro de la US.
- `category` : La catégorie dans laquelle se situe la US.
- `description` : Le texte présent sur la US et qui la définit.
- `default stars` : La valeur de la US par défaut.
- `default size` : La taille de la US par défaut.

— `[restriction]` : Une restriction de la US. Elle nécessitera le déploiement de la US d'id `restriction` avant de pouvoir être rajoutée au Sprint Backlog.

Listing B.5 – Modèle JSON de UserStory

```
{
    "id" = <int>;
    "category" = <string>;
    "description" = <string>;
    "default stars" = <int>;
    "default size" = <Size>;
    "restriction" = <int>;
}
```

Listing B.6 – Enum de Size

```
public enum Size{
        NOT_DEFINED,
        XS,
        S,
        M,
        L,
        XL
    }
```

B.3 Appels à `StateManager`

B.3.1 Menu

Nom	Pre	Post	Description
`SetDifficulty`	Difficulty	void	Applique la difficulté choisie. `Difficulty` peut prendre les valeurs `EASY`, `NORMAL` ou `HARD`.
`SetCategory`	Category	void	Applique la catégorie de US choisie. `Category` peut prendre les valeurs `GIFT_SHOP`, `DIET_COACH`, `KNOWLEDGE_MANAGEMENT` ou `CUSTOM`.
`SetGameName`	string	void	Change le nom de la partie.
`SetPokerPlanning`	bool	void	Sauvegarde si le choix d'attribuer les tâches aux US a été fait ou non.
`CreatePlayers`	List<string>	void	Crée des instances de joueurs depuis le `StateManager` sur base des surnoms entrés et les y sauvegarde.
`CreateUserStories`	Category	void	Si la catégorie est différente de `CUSTOM`, le `StateManager` crée des instances de user story de la catégorie décidée et les sauvegarde.
`SetGameState`	GameState	void	Change l'état de la partie en fonction des choix effectués. Si la catégorie de US est personnalisée, `gameState` prendra la valeur `CUSTOM_POKER_PLANNING`, sinon si l'application des tailles de US a été choisie, `gameState` prendra la valeur `POKER_PLANNING` et sinon `gameState` prendra la valeur `INITIALISATION`.

TABLE B.1 – Appels à `StateManager` depuis la scène du menu principal.

B.3.2 Attribution des tâches des US

B.3.3 Création des US

Nom	Pre	Post	Description
`GetUserStories`	/	List<UserStory>	Récupère la liste des US créées lors de la phase du menu principal.
`GetPokerPlanningState`	/	PokerPlanningState	Récupère l'état actuel de la phase d'attribution. Cet état peut être `GLOBAL`, `PRECISE` ou encore `FINISHED` lorsque la phase d'attribution est terminée.
`GetCategory`	/	Category	Récupère la catégorie de US choisie.
`SetPokerPlanningState`	PokerPlanningState	void	Change l'état courant de la scène.
`SetUserStory`	UserStory	void	Modifie une US sauvegardée.

TABLE B.2 – Appels à `StateManager` depuis la scène d'attribution de la taille des user stories

Nom	Pre	Post	Description
`GetCustomPokerPlanningState`	/	CustomPokerPlanningState	Récupère l'état actuel de la phase de création. Cet état peut être `GLOBAL`, `PRECISE` ou encore `FINISHED` lorsque la phase de création de US est terminée.
`SetGameState`	GameState	void	Change l'état courant de l'application.
`SetCustomPokerPlanningState`	CustomPokerPlanningState	void	Change l'état courant de la scène.
`SetUserStories`	List<UserStory>	void	Applique la liste des user stories générées dans la scène au `StateManager`.

TABLE B.3 – Appels à `StateManager` depuis la scène de création des user stories

B.3.4 `GameManager`

TABLE B.4 – Appels à `StateManager` depuis `GameManager`

Nom	Pre	Post	Description
`GetGameState`	/	GameState	Récupère l'état courant de la partie.
`SetGameState`	GameState	void	Modifie l'état courant de la partie.
`GetTurnState`	/	TurnState	Récupère l'état du tour du joueur courant.
`SetTurnState`	TurnState	void	Modifie l'état du tour du joueur courant.

Nom	Pre	Post	Description
`GetDifficulty`	/	Difficulty	Récupère la difficulté choisie lors de la scène du menu principal.
`GetCurrentDebt`	/	int	Récupère la valeur de la dette technique courante.
`GetDebtFactor`	/	int	Récupère le facteur de dette technique courant.
`SetStartTime`	/	DateTime	Défini le moment de départ de la partie.
`GetUserStories`	/	List<UserStory>	Récupère la liste des US instantiées en début de partie.
`SetSprintNumber`	int	void	Change le nombre de sprints effectués.
`GetCurrentDay`	/	int	Récupère le nombre du jour courant dans le sprint.
`GetTasksOnBeginSprint`	/	bool	Vérifie si l'effet permanent qui rajoute cinq tâches en début de chaque sprint est actif.
`GetOneTaskPerDay`	/	bool	Vérifie si l'effet permanent qui ajoute une tâche en début de chaque jour est actif.
`GetPlayers`	/	List<Player>	Récupère la liste des joueurs.
`GetOneMoreTaskPerRoll`	/	bool	Vérifie si l'effet permanent qui ajoute une tâche à chaque lancer de dé est actif.
`GetCurrentPlayer`	/	Player	Récupère le joueur courant.
`GetAlreadyReRoll`	/	bool	Vérifie si le dé a déjà été relancé.

Nom	Pre	Post	Description
`GetFirstTaskOrDebtChoice`	/	string	Récupère le premier choix fait pour lancer le dé. Il peut valoir "TASK" ou "DEBT".
`GetSecondTaskOrDebtChoice`	/	string	Récupère le deuxième choix fait pour lancer le dé. Il peut valoir "TASK" ou "DEBT".
`GetFirstDiceResult`	/	int	Récupère le premier résultat du dé.
`GetSecondDiceResult`	/	int	Récupère le deuxième résultat du dé.
`GetJinxed`	/	bool	Vérifie si l'effet permanent 'Poisse' est actif.
`ClearTurnState`	/	void	Réinitialise l'état lié au tour d'un joueur.
`SetSprintBeginTime`	DateTime	void	Sauvegarde le moment de départ d'un sprint.
`SetSprintStars`	int	void	Sauvegarde le nombre d'étoiles au départ d'un sprint.
`SetSprintDebt`	int	void	Sauvegarde la valeur de la dette technique au départ d'un sprint.
`SetSprintFinishedUS`	int	void	Sauvegarde le nombre de US déployées au départ d'un sprint.
`SetSprintProblemCards`	int	void	Sauvegarde le nombre de cartes 'Problème' au début d'un sprint.
`SetSprintLostTasks`	int	void	Sauvegarde le nombre de tâches perdues au début d'un sprint.
`SetLostTasks`	int	void	Sauvegarde le nombre de tâches perdues depuis le début de la partie.

Nom	Pre	Post	Description
`SetTotalTasks`	int	void	Modifie le nombre de tâches totales réalisées depuis le début de la partie.

B.3.5 `SideMenuManager`

Nom	Pre	Post	Description
`GetStartTime`	/	DateTime	Récupère le moment du début du jeu.
`GetStarsNumber`	/	int	Récupère le nombre d'étoiles accumulées depuis le début du jeu.
`GetSprintNumber`	/	int	Récupère le numéro du sprint courant.

TABLE B.5 – Appels à `StateManager` depuis le menu alternatif

B.3.6 `TDDDManager`

Nom	Pre	Post	Description
`GetUserStories`	/	List<UserStory>	Récupère la liste des US instantiées dans la partie.
`GetMaxUserStoryLowered`	/	bool	Vérifie si l'effet permanent qui diminue la taille maximum des US de 3 tâches est activé.
`GetDebtFactor`	/	int	Récupère le facteur courant de la dette technique.
`GetPlayers`	/	List<Players>	Récupère la liste des joueurs.

TABLE B.6 – Appels à `StateManager` depuis la phase d'alimentation du Sprint Backlog

B.3.7 `CardHandler`

TABLE B.7 – Appels à `StateManager` depuis `CardHandler`

Nom	Pre	Post	Description
`GetGameState`	/	GameState	Récupère l'état courant de la partie.
`SetGameState`	GameState	void	Modifie l'état courant de la partie.

Nom	Pre	Post	Description
`GetTurnState`	/	TurnState	Récupère l'état du tour du joueur courant.
`SetTurnState`	TurnState	void	Modifie l'état du tour du joueur courant.
`GetLanguage`	/	Locale	Récupère la langue choisie lors de la scène du menu principal.
`GetSkipOrDoubleDaily`	/	int	Récupère le nombre de bonus "Double la prochaine carte 'Daily' positive ou annule la prochaine carte 'Problème'" qui sont encore courant.
`SetSkipOrDoubleDaily`	int	void	Change le nombre de bonus "Double la prochaine carte 'Daily' positive ou annule la prochaine carte 'Problème'".
`GetNoMoreTestIssue`	/	bool	Vérifie si l'effet permanent qui annule les cartes 'Problème' en rapport avec des échecs de test est actif.
`SetNoMoreTestIssue`	bool	void	Change l'effet permanent qui annule les cartes 'Problème' en rapport avec des échecs de test.
`GetPlayers`	/	List<Player>	Récupère la liste des joueurs.
`GetCurrentPlayer`	/	Player	Récupère le joueur courant, et donc celui qui pioche la carte.
`GetDebtFactor`	/	int	Récupère le facteur de dette technique courant.
`GetJinxed`	/	bool	Vérifie si l'effet permanent 'Poisse' est actif.
`SetJinxed`	bool	void	Change l'effet permanent 'Poisse'.

Nom	Pre	Post	Description
`SetTasksOnBeginSprint`	bool	void	Change l'effet permanent qui rajoute cinq tâches en début de chaque sprint.
`SetMaxUserStoryLowered`	bool	void	Change l'effet permanent qui réduit le nombre de tâches maximum à réaliser pour chaque US.
`SetOneTaskPerDay`	bool	void	Change l'effet permanent qui ajoute une tâche en début de chaque jour.
`SetProblemCards`	int	void	Modifie le nombre de cartes 'Problème' piochées depuis le début de la partie.
`SetSprintProblemCards`	int	void	Modifie le nombre de cartes 'Problème' piochées depuis le début du sprint courant.

B.3.8 `ReviewManager`

Nom	Pre	Post	Description
`GetGameState`	/	GameState	Récupère l'état courant de la partie.
`SetGameState`	GameState	void	Modifie l'état courant de la partie.
`SetStarsNumber`	int	void	Modifie le nombre d'étoiles récupérées depuis le début de la partie.
`SetFinishedUS`	int	void	Modifie le nombre de US déployées depuis le début de la partie.

TABLE B.8 – Appels à `StateManager` depuis `ReviewManager`

B.3.9 `SummaryManager`

Nom	Pre	Post	Description
`GetGameState`	/	GameState	Récupère l'état courant de la partie.
`SetGameState`	GameState	void	Modifie l'état courant de la partie.
`GetSprintNumber`	/	int	Récupère le numéro du sprint courrant.
`GetSprintBeginTime`	/	DateTime	Récupère le moment du début de sprint.
`GetStarsNumber`	/	int	Récupère le nombre d'étoiles accumulées depuis le début du jeu.
`GetSprintStars`	/	int	Récupère le nombre d'étoiles accumulées au début du sprint.
`GetCurrentDebt`	/	int	Récupère la valeur de la dette technique courante.
`GetSprintDebt`	/	int	Récupère la valeur de la dette technique au début du sprint.
`GetFinishedUS`	/	int	Récupère le nombre de US déployées depuis le début du jeu.
`GetSprintFinishedUS`	/	int	Récupère le nombre de US déployées au début du sprint.
`GetSprintProblemCards`	/	int	Récupère le nombre de cartes 'Problème' piochées durant le sprint.
`GetSprintLostTasks`	/	int	Récupère le nombre de tâches qui on été perdues durant le sprint.

TABLE B.9 – Appels à `StateManager` depuis le résumé de la phase de revue

B.3.10 `EndScreenManager`

Nom	Pre	Post	Description
`GetEndTime`	/	DateTime	Récupère le moment de fin de la partie.
`SetEndTime`	DateTime	void	Sauvegarde le moment de fin de la partie.
`GetStartTime`	/	DateTime	Récupère le moment du début de la partie.
`GetPlayers`	/	List<Player>	Récupère la liste des joueurs.
`GetSprintNumber`	/	int	Récupère le nombre de sprints réalisés.
`GetCurrentDebt`	/	int	Récupère la valeur de la dette technique courante.
`GetProblemCards`	/	int	Récupère le nombre de cartes 'Problème' piochées durant toute la partie.
`GetTotalTasks`	/	int	Récupère le nombre total de tâches qui ont été ajoutées durant toute la partie.
`GetLostTasks`	/	int	Récupère le nombre de tâches qui ont été perdues durant toute la partie.
`GetDifficulty`	/	Difficulty	Récupère la difficulté choisie en début de partie.
`GetCategory`	/	Category	Récupère la catégorie de US choisie en début de partie.
`GetPokerPlanning`	/	bool	Vérifie si une phase d'attribution de taille des US a eu lieu.

TABLE B.10 – Appels à `StateManager` depuis l'écran final de la partie

B.3.11 `BurndownChartManager`

Nom	Pre	Post	Description
`GetUserStories`	/	List<UserStory>	Récupère la liste des US instantiées dans la partie.

TABLE B.11 – Appels à `StateManager` depuis le `BurndownChartManager`

B.3.12 `ScrumboardManager`

Nom	Pre	Post	Description
`GetUserStories`	/	List<UserStory>	Récupère la liste des US instantiées dans la partie.

TABLE B.12 – Appels à `StateManager` depuis le `ScrumboardManager`

Annexe C

Code

Les exécutables ainsi que l'intégralité du code se trouvent dans un Github public et libres d'accès.
Adresse du repo : `https://github.com/BellonGuillaume/Scrumble2D`

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**