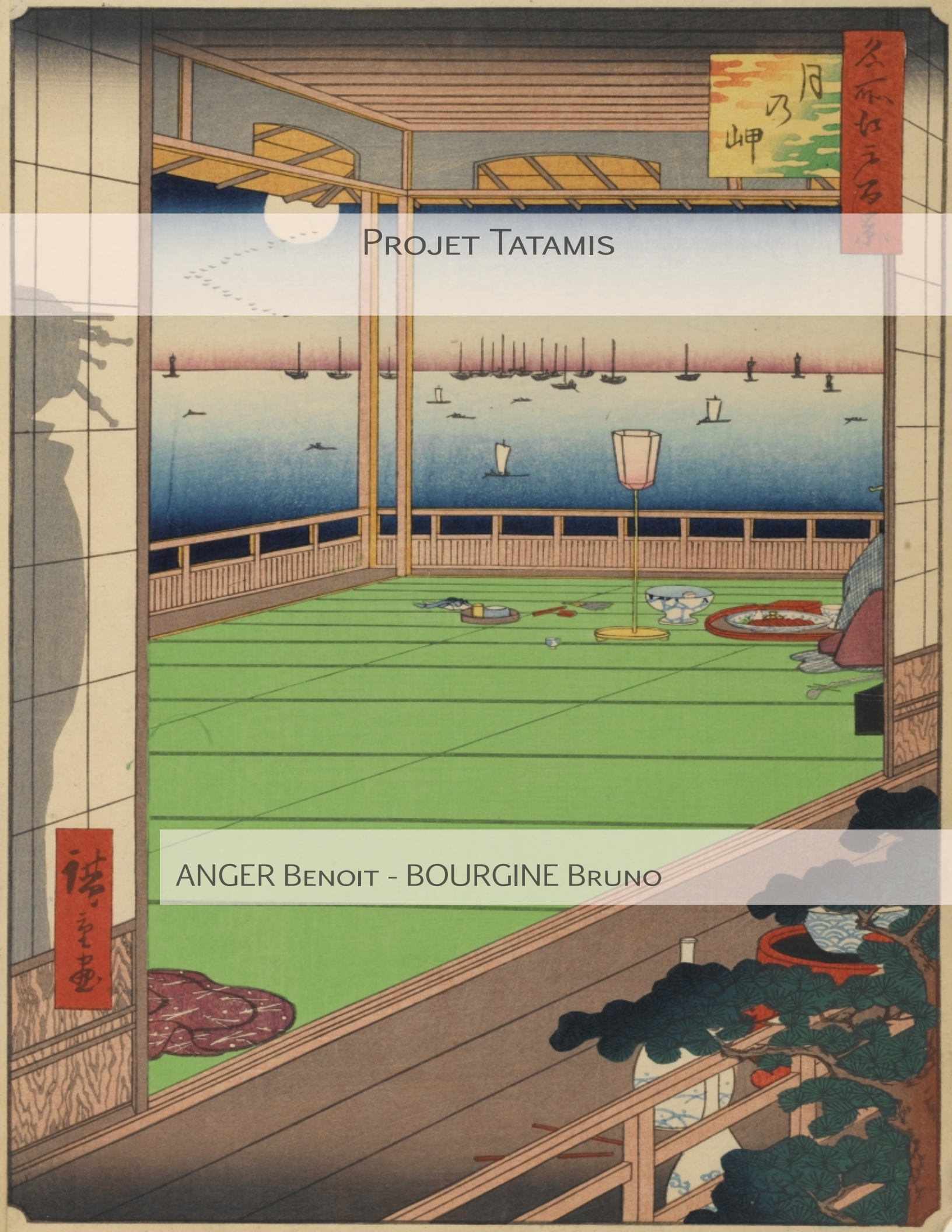
Projet Tatamis



ANGER Benoit - BOURGINE Bruno

2021-2022 ANGER Benoit - BOURGINE Bruno

Edition 0.1

Illustrations : Utagawa Hiroshige / Carrie May

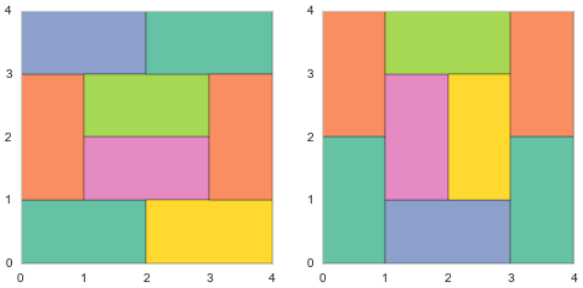
Contents

1. Présentation

Ce premier chapitre sera l'occasion d'introduire le projet tel qu'il a été initié et de détailler les outils et les méthodes envisagées pour son développement.

1.1 Descriptif de la problématique

Le pavage du plan avec des rectangle est un problème classique et déjà largement documenté, mais je souhaite l'aborder par un aspect très concret : étant donné un nombre de tatamis, quelles sont les configurations possibles.



C'est un problème que rencontre notamment toute personne qui se retrouve à devoir installer un dojo. Il existe une contrainte de base qui est que 4 tatamis ne rejoignent jamais en un même coin. Mais ont peut en ajouter d'autres : possibilité de demi-tatamis (carré), répartition des couleurs, répartition de l'usure...

1.2 Ébauche de cahier des charges

Utilisateur final : gestionnaire de dojo

L'interface utilisateur devra comprendre un menu de paramétrage basique : nombre de tatamis, contraintes géométriques, contraintes d'aspect général; ainsi qu'un affichage des dispositions envisageables.

L'utilisateur devra pouvoir saisir :

- le nombre et le type (entier/demis) de tatamis à disposition

- leurs dimensions
- leurs couleurs
- éventuellement leur état (on dispose de préférence les plus usés en périphérie)
- les dimensions du dojo

L'affichage proposera différentes disposition selon les contraintes imposées.

1.3 Membres du projet

- ANGER Benoit, étudiant licence L3 MATH-Infos
- BOULALEH Ismail, étudiant licence L3 MATH-Infos
- BOURGINE Bruno, étudiant DU CCIE

2. Cahier des charges

2.1 Démarche

La problématique initiale telle qu'elle a été énoncée est la suivante : *étant donné un nombre de tatamis, quelles sont les configurations possibles ?*

Les dojos sont de dimension m (hauteur) x n (largeur) unités. Et les tatamis sont de dimension 1 x 2 unités. Il existe parfois des demi-tatamis de dimension 1 x 1 unité. Pour disposer les tatamis, il existe une contrainte: quatre coins de tatamis ne peuvent pas se retrouver en un même point.

Nous allons ici détaillé plus précisément le cahier des charges de l'application. Nous listerons les *epics* afin d'en déduire les *users stories* et leurs tâches afférentes, et ainsi établir la *roadmap* de notre projet. Le cahier des charges est formulé du point de vue de l'utilisateur final, les *epics* étant rédigées sous la forme : *en tant que . . . ,je souhaite . . . ,afin de . . . .*

Par ailleurs afin de prioriser les demandes, nous classerons les fonctionnalités en deux catégories :

- essentielles (must have)
- optionnelles (nice to have)

2.2 Persona de l'utilisateur final

L'utilisateur final est un gestionnaire de dojo.

A propos:

- Les gestionnaires de dojo ont des profils et backgrounds variés. Il peut être très avancé ou très novice en informatique et en géométrie
- En revanche, avec un peu de formation et si on lui installe les outils, il sera capable des les utiliser même si l'ergonomie n'est pas idéale

Objectifs:

- Préparer le dojo pour qu'il soit prêt pour les entraînements et compétitions

- Assurer l'usure équitable des tatamis par des modifications régulières de disposition

Points de douleur:

- Quand il arrive sur un nouveau dojo, il est difficile de savoir rapidement si une combinaison de tatami est possible ou non
- Il est également difficile de savoir combien de tatamis lui sont nécessaires pour faire le dojo
- A chaque fois qu'il faut nettoyer le dojo, il faut enlever tous les tatamis, et il est compliqué de refaire le dojo
- En particulier lorsqu'il a des demi-tatamis ou des tatamis de couleurs différentes

2.3 Fonctionnalités essentielles

- En tant que gestionnaire de dojo, je souhaite savoir s'il existe une solution pour un dojo d'une dimension donnée, afin de savoir si je pourrais remplir mon dojo ou non.

Si une solution existe :

- *En tant que* gestionnaire de dojo, *je souhaite* connaître le nombre de tatamis nécessaires pour un dojo d'une dimension donnée, *afin de* n'en déployer que le nombre nécessaire.
- *En tant que* gestionnaire de dojo, *je souhaite* connaître le nombre de dispositions possibles pour un dojo d'une dimension donnée, *afin d'*anticiper la complexité du placement des tatamis.
- *En tant que* gestionnaire de dojo , *je souhaite* visualiser l'ensemble des dispositions possibles, pour un dojo d'une dimension donnée *afin de* m'aider à placer les tatamis sur mon dojo.
- *En tant que* gestionnaire de dojo, *je souhaite* pouvoir renseigner le nombre de tatamis dont je dispose, *afin d'*obtenir une solution adaptée à mon matériel.
- *En tant que* gestionnaire de dojo, *je souhaite* voir afficher les dimensions (longueur, largeur, surface) des dispositions proposées *afin d'* exploiter aux mieux l'espace disponible à l'intérieur et à l'extérieur du tatamis.

2.4 Fonctionnalités optionnelles

- *En tant que* gestionnaire de dojo, *je souhaite* connaître le nombre de dispositions possibles, modulo une rotation ou une symétrie *afin de* ne voir sur l'écran que les solutions réellement différentes.
- *En tant que* gestionnaire de dojo, *je souhaite* visualiser l'ensemble des dispositions possibles, modulo une rotation ou une symétrie *afin de* ne voir sur l'écran que les solutions réellement différentes.
- *En tant que* gestionnaire de dojo, *je souhaite* pouvoir modifier les dimensions d'un tatamis *afin de* obtenir des propositions correspondant à mon matériel.

- *En tant que* gestionnaire de dojo, *je souhaite* pouvoir créer des catégories de couleurs de tatamis *afin de* visualiser des propositions de placement avec les couleurs réelles.

3. Méthodologie

3.1 Choix de méthodologie

La méthodologie agile nous paraît très adaptée au développement de notre programme. En effet, la méthodologie agile:

- Est particulièrement adaptée à la résolution de problème complexes et incertaines ou l'on ne sait pas forcément avec précisions l'objectif final ou la manière d'y arriver, ce qui est notre cas
- Est basée sur l'itération avec la production de livrables testables à la fin de chaque *sprint*, ce qui semble adapté pour produire nos différentes versions (alpha, beta...)
- Est basée sur une équipe multidisciplinaire qui couvre toutes les compétences pour produire un produit fini et qui s'auto organise, ce qui parait également adapté à notre contexte

La méthodologie agile, étant particulièrement adaptée aux situations d'incertitude, préconise une planification au fur et à mesure de temps, plutôt que d'importantes et lourdes activités de planification en début de projet car les informations manquent pour cette planification totale en amont.

Pour nous donner un cadre, nous nous inspirons très fortement du schéma "Scrum" et du guide Scrum[1], tout en l'adaptant à notre situation présente avec des ressources limitées et des contraintes particulières.

3.2 Les événements

3.2.1 Les sprints et les phases du projet

Les sprints sont des périodes de développement ayant un objectif précis et permettant d'arriver à une version du programme. Compte tenu du planning imposé par l'exercice, les sprints seront de durée variable et d'une durée légèrement supérieur à un mois (contrairement à ce qui est suggéré par le schéma Scrum).

[1] The Scrum Guide, Ken Schwaber et Jeff Sutherland, Novembre 2017 https://www.scrum.org/resources/scrum-guide

Les sprints commencent par la session de planning et se terminent par la revue et la rétrospective (événements détaillés ci-après). Elles comprennent également des activités de 'raffinement' ou de préparation du prochain sprint, pour qu'un nouveau sprint puisse commencer immédiatement après la clôture du précédent sprint.

Quatre sprints sont programmés pour le projet aboutissant aux versions Alpha, Beta, Release Candidate et Production.

Nb: Une phase additionnelle de pré-développement aura lieu en amont pour la préparation du projet, mais est organisée de manière ad-hoc et ne peut être considérée comme un sprint. Cette phase a pour objectif d'analyser la demande (le cahier des charges), de déterminer la méthodologie et gouvernance et de préparer le développement pour aboutir sur une roadmap, un plan de développement global du programme qui sera bien sûr affiné au cours du temps.

3.2.2 La session de planning du sprint

Pour chaque sprint la session de planning permet de déterminer:

- Le *Quoi*: quels éléments du backlog global peuvent être embarqué dans ce sprint pour créer le Sprint backlog
- Le *Comment*: comment chaque élément du Sprint backlog seront techniquement traités
- Le *Pourquoi*: quel objectif pour le sprint, sachant que chaque sprint doit délivrer un produit qui peut être limité en fonctionnalités mais qui fonctionne

Les décisions sont prises de la manière suivante:

- Quoi et pourquoi :

  Les propositions d'éléments à ajouter et d'objectif du sprint viennent du product owner. L'équipe de développement prend ensuite la décision de manière souveraine et autonome en session de planification.

- Comment :

  Chaque élément du Sprint backlog est discuté techniquement pour le décomposer en plus petites tâches qui feront l'objet de tickets.

  En cas de manque de compétences techniques, des tickets sont prévus pour la réalisation de recherches.

  Cette session couvre également les questions d'architecture du programme: choix du nombre de classes, de leurs interactions...

3.2.3 "L'hebdo"

Compte tenu de la situation particulière, un point de contact quotidien comme préconisé par le schéma Scrum n'est pas envisageable. Il sera remplacé par:

- Un point hebdomadaire facilité par le Scrum master pour un focus particulier sur l'échange, les challenges et solutions.
- La revue des tâches et le statut sont quand à eux discutés à travers:

  - Une communication continue sur Slack
  - Une mise à jour en direct des avancées sur Redmine, directement sur les tâches. Redmine apportant la visibilité nécessaire à tous les membres de l'équipe pour comprendre le statut rapidement grâce à la combinaison du diagramme Gantt, d'un tableau Kanban des tâches et d'un tableau de roadmap qui suit le pourcentage de complétude du backlog du sprint

3.2.4 La revue du sprint

Chaque sprint se termine par une revue du produit livré à la fin du sprint. Les fonctionnalités développées sont discutées, ainsi que les challenges rencontrés. L'équipe commence à se projeter également sur le prochain sprint et ce qu'il faut faire ensuite.

3.2.5 La retrospective

L'objectif de cette réunion est une introspection pour une amélioration continue notamment de la collaboration au sein de l'équipe, les processus et les outils. La session est facilitée par le Scrum master et se déroule selon les principes suivants:

- Discussions autour de 3 blocs successivement:
  - *Garde*: ce que l'on considère adapté et à conserver dans le futur
  - *Start*: ce que l'on souhaite commencer à faire pour améliorer la situation
  - *Stop*: ce que l'on souhaite arrêter sur un constat d'échec
- Étapes de chaque blocs:
  - Réflexion individuelle de points/idées à ajouter pour le bloc discute
  - Discussion de groupe pour regrouper les points mentionnés par thème
  - Résumé des actions/idées retenues
- La session se termine par la détermination du plan d'amélioration regroupant les idées retenues et en ajoutant une composante de temps et responsabilité des actions
- Outil utilisé pour la session: Miro (outil interactif et collaboratif permettant notamment de créer et arranger facilement des "post it" représentant les points/idées)

3.3 Les éléments de formalisation

3.3.1 Cahier des charges global (et les Epics)

Le cahier des charges global représente la demande et le besoin de l'utilisateur. Il comprend une description du/des type d'utilisateur(s). Il est constitué des fonctionnalités majeures que l'utilisateur souhaite pouvoir effectuer grâce au programme.

Il est constitué de quelques "Epics" qui expliquent le type d'utilisateur, l'objectif et la raison. Les epics sont rédigées sous la forme suivante:

"En tant que . . . , je souhaite . . . , afin que . . . "

Les épics ont les statuts suivants Statut: Backlog (non planifié), À commencer, En cours, Achevée, Rejeté.

> *Définition d'achèvement ('Definition of Done') des epics :*
> *Pour assurer une transparence et que toutes les parties prenantes aient la même définition, une définition d'achèvement est formalisée ainsi:*
>
> 1. *Toutes les users stories de l' epic sont achevés*
> 2. *Les activités de refactoring ont été réalisées (pour vérification de la concordance avec les principes de développement SOLID)*
> 3. *Les tests utilisateurs sont réalisés et leur résultat est positif*

3.3.2 Le backlog global (et les User Stories)

Le backlog est constituée de l'ensemble d'éléments qui pourraient être construits (codés) pour arriver à un produit fini idéal. Il représente des besoins très spécifiques des utilisateurs.

Il est constitué de *User stories* (scénarios utilisateur) qui expliquent la même chose et sont rédigées de la même manière que les epics, mais qui représentent de plus petits éléments.

Une *User story* représente une fonctionnalité très particulière et sa structure est la suivante:

- Nom de la fonctionnalité
- Contexte
- Utilisateur ("En tant que"), Objectif ("je souhaite"), Raison ("afin que")
- Test utilisateur d'acceptance (plus d'explication à la suite)
- Statut: Backlog (non planifié), À commencer, En cours, Achevée, Rejeté
- Sprint de rattachement (ou Backlog global si pas encore planifié au sein d'un sprint)
- Priorisation (proposition de sprint)
- Documentation utilisateur
- Vérification de complétude (voir page 16 la définition d'achèvement)

Les user stories suivent le principe *INVEST*:

- *Independant* (Indépendante) : chaque user story est indépendante des autres
- *Negotiable* (Négociable) : une user story décrit un besoin; la manière d'y répondre reste négociable
- *Valuable* (Apporte de la valeur) : chaque user story doit apporter de la valeur à l'utilisateur
- *Estimable* (Estimable) : une user story doit être estimable par l'équipe de développement, c'est a dire que l'équipe de développement doit avoir assez d'information pour comprendre l'effort nécessaire à la mise en oeuvre

- *Small* (Petit) : une user story doit être petite, c'est à dire traitable en quelques jours (difference avec un Epic)

- *Testable* (Testable) : une user story doit pouvoir être testable, ce qui permet de s'assurer qu'elle est assez bien définie.

Pour assurer la cohérence avec le modèle INVEST, et notamment la valeur, l'estimabilité et la testabilité, les tests unitaires d'acceptance sont décrits pour chaque user story et sont notamment un des paramètres de la définition d'achèvement.

> *Définition d'achèvement ('Definition of Done') des user stories :*
> *Pour assurer une transparence et que toutes les parties prenantes aient la même définition, une définition d'achèvement est formalisée ainsi:*
>
> 1. *Toutes les tâches des users stories sont achevés*
> 2. *Les test utilisateur d'acceptance sont réalisés et leur résultat est positif (en cas de bugs, ils ont étés corrigés)*
> 3. *La documentation utilisateur est terminée*

Les User stories peuvent être créées à tout moment du projet. Elles commencent toujours par être ajoutées au Backlog global, notamment lors des activités de 'raffinement' en préparation du prochain sprint. Elles sont ensuite ajoutées au Backlog d'un sprint particulier en session de planification. Une partie importante des User stories sera créer en phase de pré-développement, mais de nombreuses User stories seront également créer au fur et à mesure du temps et que le développement avance.

Le backlog global est ordonné par le Product Owner, c'est-à- dire que le Product Owner propose les tâches à embarquer pour chaque sprint en session de planification.

3.3.3 Roadmap globale

La 'roadmap' (feuille de route) globale est un plan de développement du programme, pour arriver au produit fini. Elle reprend les Epics et User stories de chacune des étapes pour arriver au produit final:

- Pre-développement

- Versions successives: Alpha, Beta, Release Candidate, Production

La roadmap peut également en vue très détaillée reprendre les tâches de chaque sprint.

Comme expliqué plus haut, en méthodologie agile, la roadmap est vivante et se construit au fur et à mesure du temps, comme expliqué plus haut. Un premier jet en lance lors de la phase de pré-développement mais c'est un travail continue et la roadmap est affinée en permanence, notamment par l'ajout de User stories ou de tâches permettant d'arriver à l'objectif.

L'objectif est d'éviter les tâches de planification trop lourdes et trop incertaines en amont, alors que l'incertitude est forte, et de planifier plutôt au fil du temps dans des conditions de meilleure connaissance.

Ainsi, plusieurs versions de la roadmap seront présentées au cours du projet.

3.3.4 Le backlog d'un sprint (et les Tâches)

Le backlog d'un sprint est le plan de développement d'un sprint et est le résultat des discussions de la session de planification. Il comprend:

- La liste des user stories qui seront traitées dans le sprint
- La décomposition des user story en une liste des tâches par user story pour la réaliser

Une tâche est un petit élément de code, une pièce du puzzle pour arriver à réaliser la user story dans son ensemble. Toute tâche comprend une documentation technique et une définition des tests d'acceptance unitaire, pour permettre au développeur de bien comprendre le résultat attendu.

Définition d'achèvement ('Definition of Done') des tâches :
Pour assurer une transparence et que toutes les parties prenantes aient la même définition, une définition d'achèvement est formalisée ainsi:

1. *Le code est écrit*
2. *Le code a été revu par un autre développeur*
3. *Les test unitaires sont réalisés et leur résultat est positif (en cas de bugs, ils ont étés corrigés)*
4. *La documentation utilisateur est terminée*
5. *La documentation technique a été revue par un autre développeur*

3.4 Les rôles

3.4.1 Description des rôles

- **Le 'product owner'**

  Il est responsable de maximiser la valeur créée par le programme résultant du travail de l'équipe. Ses tâches principales consistent en:

  - Exprimer les éléments du Backlog
  - Ordonner le Backlog par ordre de priorité
  - Assurer la clarté du backlog et que ces éléments soient bien compris de tous

- **Les membres de l'équipe de développement**

  Ils sont responsables de la création du produit à l'issue de chaque sprint. Ils s'auto-organisent et n'ont pas de titre ou hiérarchie au sein de l'équipe

- **Le 'scrum master'**

  Il aide et supporte l'utilisation de la méthodologie scrum. Ses tâches principales consistent en:

  - Aider le product owner dans la gestion du backlog, notamment avec les techniques adaptées

- Coache les membres de l'équipe de développement pour s'auto-organiser et intervient en cas de blocage
- Interagit avec le reste de l'organisation (par exemple le Big Boss et le Directeur Technique) pour que l'équipe conserve son focus sur le développement

3.4.2 Assignation des rôles

Compte tenu de la taille réduite de l'équipe, les membres peuvent être amenés à jouer plusieurs rôles. L'organisation de l'équipe sera la suivante:

Titre	Rôle au sein de l'équipe Scrum	Nom
Big Boss		Tristan COLOMBO
Directeur Technique		Tristan COLOMBO
Chef de projet	Membre de l'équipe de développement	Bruno BOURGINE
Développeur	Scrum master et Product Owner et membre de l'équipe de développement	Benoit ANGER
Développeur	Membre de l'équipe de développement	Ismail BOULALEH

3.5 L'approche de test

L'approche de test sera à niveaux multiples:

- Test unitaires pour les tâches
- Test utilisateurs pour les Epics et User stories

Dans tous les cas, les tests sont définis en amont, au moment de la rédaction des epics, user stories et tâches. L'objectif est de:

1. permettre au développeur de bien comprendre le résultat attendu tout en lui laissant la liberté pour l'atteindre et
2. d'apporter une transparence à toutes les parties prenantes.

4. Choix des outils et langages

4.1 Outils de communications

4.1.1 Redmine

Cet outil de gestion de projet a été choisi en premier lieu pour sa disponibilité immédiate (il n'y a pas eu d'installation ou de paramétrage de serveur à réaliser) mais aussi pour sa complétude en terme d'outils. Il dispose en effet de l'ensemble des fonctionnalités dont nous avions besoin pour ce qui est de la création, de l'ordonnancement et du suivi des demandes. En cela il est tout à fait adapté à la méthodologie choisie.

Nous avons pu le paramétrer un peu plus finement de façon à ce que les caractéristiques et l'évolution des demandes correspondent à la terminologie employée pour détailler notre projet : type de tracker, statut des demandes, champs personnalisés...

4.1.2 Slack

Slack a été choisi comme outil de communication entre les membres de l'équipe afin d'établir des canaux de discussion différenciés. Cela permet à l'équipe des échanges plus ciblés et donc plus efficaces, ainsi qu'une vision plus ordonnée de l'historique des communications.

4.1.3 Github

La plateforme Github a été choisie pour héberger et gérer l'ensemble des éléments du projet, à savoir le code de l'application ainsi que le rapport.

Le dépot est consultable à l'adresse : `https://github.com/bubobou/tatamis`

4.2 Développement

4.2.1 Langage

De part sa facilité d'assimilation et les nombreuses bibliothèques disponibles, le choix du langage de programmation s'est porté sur Python dans sa version 3.9.

4.2.2 Bibliothèques

Différentes bibliothèques nous ont parues d'emblée utiles pour aborder ce projet. Tout d'abord une recherche documentaire nous a menée vers la bibliothèque facile permettant de traiter de la programmation par contrainte. Même s'il ne sera pas forcément retenue dans la version finale, sa disponibilité nous permet d'aborder plus sereinement notre problématique.

Par ailleurs dans la finalité d'une application avec interface graphique, potentiellement portée sur terminal mobile, nous avons envisagé l'utilisation de bibliothèques et d'utilitaires tels que :

- PyQt
- Kivy
- python for android

5. Roadmap globale

5.1 Roadmap au 08/01/2022

5.1.1 Éléments

Préparation

Échéance dans un jour (13/01/2022)

100%

19 demandes (19 fermées — 0 ouverte)

Demandes liées

~~Phase de pré développement #475~~: US Établir la roadmap
~~Phase de pré développement #477~~: TACHE Recherche de la documentation existante
~~Phase de pré développement #487~~: EPIC Préparer les développements
~~Phase de pré développement #488~~: US Choisir et mettre en place les outils
~~Phase de pré développement #489~~: TACHE Choix des outils
~~Phase de pré développement #490~~: TACHE Configuration des outils - Redmine (en accordance avec la méthodologie)
~~Phase de pré développement #491~~: TACHE Configuration des outils - Slack
~~Phase de pré développement #492~~: TACHE Configuration des outils - Github (y compris integration a Slack si possible)
~~Phase de pré développement #493~~: US Choisir et développer la méthodologie de travail
~~Phase de pré développement #494~~: TACHE Discussion et choix de méthodologie
~~Phase de pré développement #495~~: TACHE Formalisation de la méthodologie
~~Phase de pré développement #500~~: TACHE Relecture et commentaires sur la formalisation de la méthodologie
~~Phase de pré développement #501~~: US Analyser le cahier des charges
~~Phase de pré développement #502~~: TACHE Formalisation du cahier des charges
~~Phase de pré développement #503~~: TACHE Analyse du cahier des charges
~~Phase de pré développement #504~~: US Revoir la littérature/documentation existante
~~Phase de pré développement #505~~: TACHE Analyse de la littérature
~~Phase de pré développement #506~~: TACHE Choix des outils (bibliotheques)
~~Phase de pré développement #507~~: TACHE Formaliser de la Roadmap global / plan de développement du programme

Alpha

1%

12 demandes (0 fermée — 12 ouvertes)

Demandes liées

- Sprint backlog Alpha #480: US Calculer le nombre de tatamis nécessaires
- Sprint backlog Alpha #508: EPIC Développer pour un gestionnaire de dojo un outil simple lui permettant de saisir les dimensions du dojo et de savoir si un solution existe, le nombre de tatamis necessaires et le nombre de dispositions possibles
- Sprint backlog Alpha #509: US Créer la fonction permettant de calculer le nombre de solutions au problème étant donné les dimensions du dojo
- Sprint backlog Alpha #510: US Comprendre si une solution existe
- Sprint backlog Alpha #511: TACHE Demander les dimensions et permettre au gestionnaire de dojo de les saisir
- Sprint backlog Alpha #512: TACHE Utiliser la fonction pour renvoyer la réponse au gestionnaire de dojo
- Sprint backlog Alpha #513: US Comprendre le nombre de dispositions possibles
- Sprint backlog Alpha #514: TACHE Permettre au gestionnaire de dojo de poser cette question
- Sprint backlog Alpha #515: TACHE Utiliser la fonction pour renvoyer la réponse au gestionnaire de dojo
- Sprint backlog Alpha #516: TACHE Permettre au gestionnaire de dojo de poser cette question
- Sprint backlog Alpha #517: TACHE Créer une fonction permettant de calculer le nombre de tatamis nécessaires étant donné des dimensions de dojo et qu'une solution existe
- Sprint backlog Alpha #518: TACHE Utiliser la fonction pour renvoyer la réponse au gestionnaire de dojo

Beta

0%

3 demandes (0 fermée — 3 ouvertes)

Demandes liées

- Sprint backlog Beta #476: US Afficher visuellement une disposition possible
- Sprint backlog Beta #478: US Afficher visuellement toutes les dispositions possibles
- Sprint backlog Beta #519: EPIC Permettre au gestionnaire de dojo de visualiser les solutions

Pre-Prod / Release candidate

0%

3 demandes (0 fermée — 3 ouvertes)

Demandes liées

- Sprint backlog RC #483: EPIC Intégrer l'utilisation des demi-tatamis dans le problème et la solution
- Sprint backlog RC #484: US Comprendre s'il est utile d'utiliser des demi-tatamis pour trouver une solution
- Sprint backlog RC #520: US Afficher visuellement toutes les dispositions possibles avec des demi-tatamis

Production

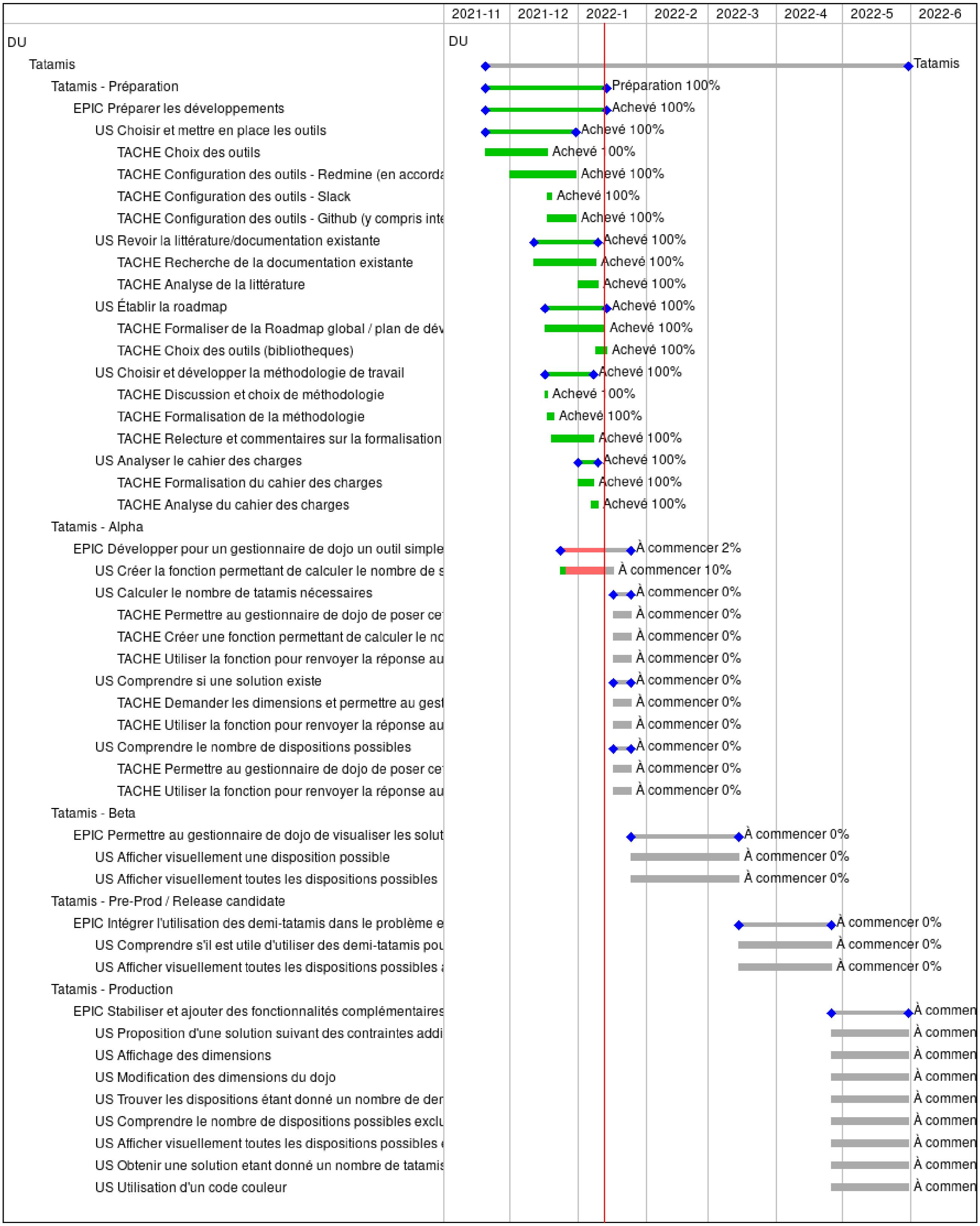
0%

9 demandes (0 fermée — 9 ouvertes)

Demandes liées

- Sprint backlog Production #479: US Proposition d'une solution suivant des contraintes additionnelles
- Sprint backlog Production #481: US Affichage des dimensions
- Sprint backlog Production #482: US Modification des dimensions du dojo
- Sprint backlog Production #521: EPIC Stabiliser et ajouter des fonctionnalités complémentaires
- Sprint backlog Production #522: US Trouver les dispositions étant donné un nombre de demi-tatamis
- Sprint backlog Production #523: US Comprendre le nombre de dispositions possibles excluant les dispositions symétriques
- Sprint backlog Production #524: US Afficher visuellement toutes les dispositions possibles excluant les dispositions symétriques
- Sprint backlog Production #525: US Obtenir une solution etant donné un nombre de tatamis
- Sprint backlog Production #526: US Utilisation d'un code couleur

5.1.2 Gantt

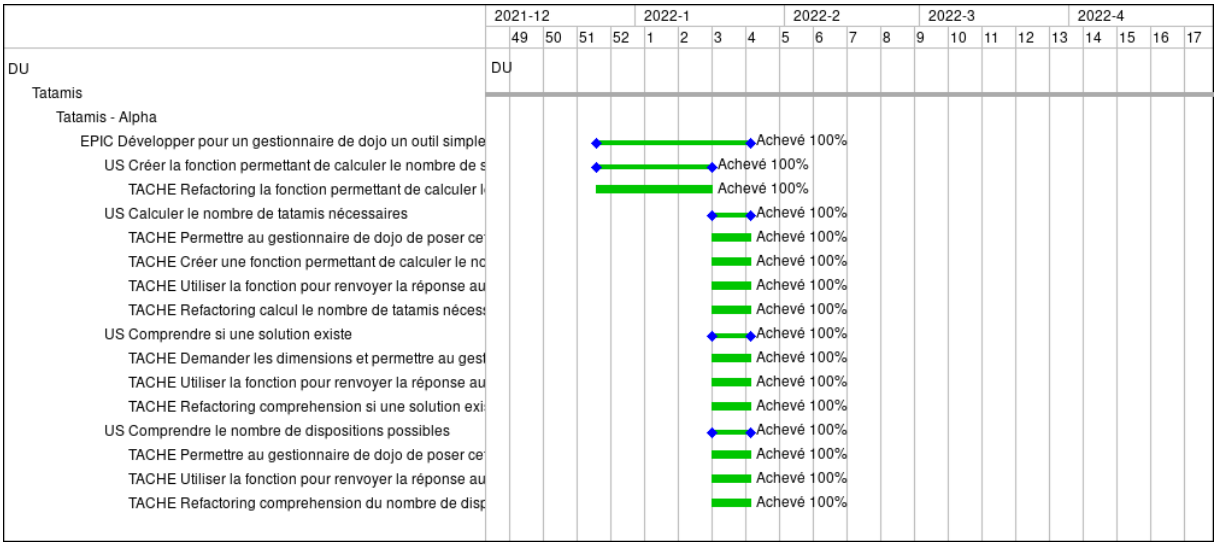

2021-11
2021-12
2022-1
2022-2
2022-3
2022-4
2022-5
2022-6
DU
DU
Tatamis
Tatamis
Tatamis - Préparation
Préparation 100%
EPIC Préparer les développements
Achevé 100%
US Choisir et mettre en place les outils
Achevé 100%
TACHE Choix des outils
Achevé 100%
TACHE Configuration des outils - Redmine (en accorda
Achevé 100%
TACHE Configuration des outils - Slack
Achevé 100%
TACHE Configuration des outils - Github (y compris inte
Achevé 100%
US Revoir la littérature/documentation existante
Achevé 100%
TACHE Recherche de la documentation existante
Achevé 100%
TACHE Analyse de la littérature
Achevé 100%
US Établir la roadmap
Achevé 100%
TACHE Formaliser de la Roadmap global / plan de dév
Achevé 100%
TACHE Choix des outils (bibliotheques)
Achevé 100%
US Choisir et développer la méthodologie de travail
Achevé 100%
TACHE Discussion et choix de méthodologie
Achevé 100%
TACHE Formalisation de la méthodologie
Achevé 100%
TACHE Relecture et commentaires sur la formalisation
Achevé 100%
US Analyser le cahier des charges
Achevé 100%
TACHE Formalisation du cahier des charges
Achevé 100%
TACHE Analyse du cahier des charges
Achevé 100%
Tatamis - Alpha
EPIC Développer pour un gestionnaire de dojo un outil simple
À commencer 2%
US Créer la fonction permettant de calculer le nombre de s
À commencer 10%
US Calculer le nombre de tatamis nécessaires
À commencer 0%
TACHE Permettre au gestionnaire de dojo de poser ce
À commencer 0%
TACHE Créer une fonction permettant de calculer le nc
À commencer 0%
TACHE Utiliser la fonction pour renvoyer la réponse au
À commencer 0%
US Comprendre si une solution existe
À commencer 0%
TACHE Demander les dimensions et permettre au gest
À commencer 0%
TACHE Utiliser la fonction pour renvoyer la réponse au
À commencer 0%
US Comprendre le nombre de dispositions possibles
À commencer 0%
TACHE Permettre au gestionnaire de dojo de poser ce
À commencer 0%
TACHE Utiliser la fonction pour renvoyer la réponse au
À commencer 0%
Tatamis - Beta
EPIC Permettre au gestionnaire de dojo de visualiser les solut
À commencer 0%
US Afficher visuellement une disposition possible
À commencer 0%
US Afficher visuellement toutes les dispositions possibles
À commencer 0%
Tatamis - Pre-Prod / Release candidate
EPIC Intégrer l'utilisation des demi-tatamis dans le problème e
À commencer 0%
US Comprendre s'il est utile d'utiliser des demi-tatamis pou
À commencer 0%
US Afficher visuellement toutes les dispositions possibles a
À commencer 0%
Tatamis - Production
EPIC Stabiliser et ajouter des fonctionnalités complémentaires
À commen
US Proposition d'une solution suivant des contraintes addi
À commen
US Affichage des dimensions
À commen
US Modification des dimensions du dojo
À commen
US Trouver les dispositions étant donné un nombre de der
À commen
US Comprendre le nombre de dispositions possibles exclu
À commen
US Afficher visuellement toutes les dispositions possibles e
À commen
US Obtenir une solution etant donné un nombre de tatamis
À commen
US Utilisation d'un code couleur
À commen

6. Version Alpha

6.1 Backlog/roadmap du sprint Alpha

L'objectif de ce sprint est de poser les fondations et délivrer les fonctionnalités de bases attendues par l'utilisateur, c'est-à- dire la compréhension des dispositions possibles et impossibles.

Pour ce faire le backlog suivant a été "embarqué" dans cette version et a résulté en la roadmap suivante:



6.2 Tests

id	Sujet	Test d'acceptance	Méthode de test	Résultat
518	TACHE Utiliser la fonction pour renvoyer la réponse au gestionnaire de dojo	Quand le programme est lancé et que l'utilisateur saisi comme attendu, une réponse lui est renvoyée dans la console	Manuel	OK
517	TACHE Créer une fonction permettant de calculer le nombre de tatamis nécessaires étant donné des dimensions de dojo et qu'une solution existe	Quand la fonction est exécutée, elle retourne le nombre de tatamis nécessaires	Automatisé	OK
516	TACHE Permettre au gestionnaire de dojo de poser cette question	1. Quand le programme est lancé, il demande a l'utilisateur la largeur du dojo	Automatisé	OK
		2. Quand le programme est lancé, il demande a l'utilisateur la longueur du dojo	Automatisé	OK
		3. Quand le programme est lancé, une option est disponible pour l'utilisateur pour poser cette question.	Automatisé	OK

id	Sujet	Test d'acceptance	Méthode de test	Résultat
515	TACHE Créer une fonction permettant de calculer le nombre de tatamis nécessaires étant donné des dimensions de dojo et qu'une solution existe	Quand la fonction est executée, elle retourne le nombre de tatamis nécessaires	Automatisé	OK
514	TACHE Permettre au gestionnaire de dojo de poser cette question	1. Quand le programme est lancé, il demande à l'utilisateur la largeur du dojo	Automatisé	OK
		2. Quand le programme est lancé, il demande à l'utilisateur la longueur du dojo	Automatisé	OK
		3. Quand le programme est lancé, une option est disponible pour l'utilisateur pour poser cette question.	Automatisé	OK
513	US Comprendre le nombre de dispositions possibles	1. Étant donné que l'utilisateur saisie des dimensions de dojo, quand il saisie 2, il obtient le nombre conformément à l'article de Ruskey et Woodcock de 2009	Automatisé	OK
		2. Étant donné que l'utilisateur saisie des dimensions de dojo en inversant la longueur et la largeur, quand il saisie 2, il obtient le nombre conformement a l'article de Ruskey et Woodcock de 2009	Automatisé	OK
512	TACHE Utiliser la fonction pour renvoyer la réponse au gestionnaire de dojo Alpha	Quand le programme est lancé et que l'utilisateur saisie comme attendu, une réponse lui est renvoyée dans la console	Automatisé	OK
511	TACHE Demander les dimensions et permettre au gestionnaire de dojo de les saisir	1. Quand le programme est lancée, il demande a l'utilisateur la largeur du dojo	Automatisé	OK
		2. Quand le programme est lancé, il demande a l'utilisateur la longueur du dojo	Automatisé	OK
510	US Comprendre si une solution existe	1. Etant donné que l'utilisateur saisie des dimensions de dojo n'ayant pas de solution, quand il saisie 1, il obtient 'Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo'	Automatisé	OK
		2. Étant donné que l'utilisateur saisie des dimensions de dojo ayant au moins une disposition, quand il saisie 1, il obtient 'Il existe au moins une disposition avec des tatamis 2x1 pour ce dojo'	Automatisé	OK

id	Sujet	Test d'acceptance	Méthode de test	Résultat
509	US Créer la fonction permettant de calculer le nombre de solutions au problème étant donné les dimensions du dojo	1. Quand la fonction est exécutée, elle retourne le nombre de dispositions possibles conformément a l'article de Ruskey et Woodcock de 2009	Automatisé	OK
		2. Quand la fonction est exécutée en inversant la longueur et la largeur, elle retourne le nombre de dispositions possibles conformément a l'article de Ruskey et Woodcock de 2009	Automatisé	OK
509	EPIC Développer pour un gestionnaire de dojo un outil simple lui permettant de saisir les dimensions du dojo et de savoir si un solution existe, le nombre de tatamis nécessaires et le nombre de dispositions possibles	cf. tests utilisateurs des US		
480	US Calculer le nombre de tatamis nécessaires	1. Étant donne que l'utilisateur saisit des dimensions de dojo avec une solution qui existe, quand il saisie 3, il obtient le nombre de tatamis nécessaires	Automatisé	OK
		2. Étant donne que l'utilisateur saisit des dimensions de dojo avec aucune disposition possible, quand il saisie 3, il obtient un message indiquant l'absence de solution	Automatisé	OK

Les tests automatisés sont définis dans le fichier `test_alpha.py` et `test_interface.py`. Ils peuvent être reproduits en l'exécutant avec ligne de commande `python3 -m pytest`.

6.3 Documentation utilisateur Alpha

6.3.1 Prérequis

Configuration et installations requises:

- Python 3.9 ou supérieur
- Librairies Python: aucune

6.3.2 Comment trouver le nombre de tatamis nécessaires pour remplir un dojo?

1. Lancer le programme dans un Terminal (ligne de commande: `python3 interface.py`)
2. Entrer (dans le Terminal) la longueur et la largeur du dojo en suivant les questions posées par le programme. Nb: *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
3. Répondre "3" à la question du programme "Que cherchez vous?"
4. Interpréter la réponse:
   (a) **Réponse** : "Le nombre de tatamis 2x1 nécessaires pour ce dojo est : [Nombre]". **Interprétation**: il existe au moins une disposition possible et vous aurez besoin d'exactement [Nombre] tatamis pour remplir le dojo.
   (b) **Réponse** : "Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo". **Interprétation**: il n'existe aucune disposition possible de tatamis 2 x 1 pour remplir le dojo.

6.3.3 Comment savoir s'il existe une disposition possible de tatamis pour un dojo donné?

1. Lancer le programme dans un Terminal (ligne de commande: `python3 interface.py`)
2. Entrer (dans le Terminal) la longueur et la largeur du dojo en suivant les questions posées par le programme. Nb: *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
3. Répondre "1" à la question du programme "Que cherchez vous?"
4. Interpréter la réponse:
   (a) **Réponse** : "Il existe au moins une disposition avec des tatamis 2x1 pour ce dojo". **Interprétation**: il existe au moins une disposition possible.
   (b) **Réponse** : "Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo". **Interprétation**: il n'existe aucune disposition possible de tatamis 2 x 1 pour remplir le dojo.

6.3.4 Comment savoir combien il existe de dispositions possibles de tatamis pour un dojo donné?

1. Lancer le programme dans un Terminal (ligne de commande: `python3 interface.py`)
2. Entrer (dans le Terminal) la longueur et la largeur du dojo en suivant les questions posées par le programme. Nb: *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
3. Répondre "2" à la question du programme "Que cherchez vous?"
4. Interpréter la réponse:
   (a) **Réponse** : "Il existe [Nombre] dispositions possibles". **Interprétation**: des dispositions existent pour ce dojo et il y en a [Nombre].
   (b) **Réponse** : "Il existe 0 disposition possible". **Interprétation**: il n'existe pas de disposition pour ce dojo.

6.4 Explication des algorithmes et choix de programmation

6.4.1 Algorithme pour les calculs de nombre de dispositions

Nous avons tenté deux approches pour répondre à ces fonctionnalités de base:

1. Exploitation de la bibliothèque facile écrite en python pour résoudre des problèmes en programmation par contrainte. La publication de Xavier Olive[1] proposant une application de cette bibliothèque au problème qui nous concerne,nous avons explorer la possibilité d'adapter le programme proposé dans la publication. Si le nombre de dispositions proposées lors des calculs pour des dimensions de dojo données est tout à fait cohérent avec les valeurs trouvées dans les autres publications traitant de ce problème, il nous est rapidement apparu que les dimensions étaient limités, et que le temps de calcul n'était pas satisfaisant.

[1] `https://www.xoolive.org/2016/02/29/pavage-par-tatamis.html`

2. Application de l'approche proposée par Dean Hickerson[2]: nous avons codé le programme mathématique qu'il décrit en python.

   La méthode 2 a finalement été retenue car elle a l'avantage d'être très légère et rapide en exécution. Elle peut notamment calculer rapidement des réponses même si les dimensions sont grandes (nous n'avons d'ailleurs par limites les dimensions saisies à un certain nombre).

6.4.2 Choix de programmation Interface

Pour faire à l'essentiel, il a été choisi d'utiliser le terminal pour les interactions entre l'utilisateur et le programme. Ce n'est évidemment pas idéal, mais cela a permis de rapidement traiter la partie interface pour se concentrer sur les fonctionnalités et les calculs mathématiques permettant d'y répondre.

6.4.3 Choix de la structure du programme

Il nous semble important de suivre et mettre en application des principes *SOLID*, et en particulier du *Single Responsibility Principle* et *Open-Closed Principle*. A ce stade, vu la simplicité du programme, ce n'est pas forcément nécessaire ni applicable de manière évidente, mais nous souhaitons construire des fondations solides pour la suite:

- Un fichier a une fonction principale

  Dans cette version alpha, nous avons 2 fichiers (un fichier de front-end `interface.py` et un fichier de back-end `alpha.py`) :

  - `interface.py` : fichier contient toutes les fonctionnalités de front-end, c'est à dire l'interface utilisateur
  - `alpha.py` : fichier qui reprend les fonctions de calculs (basiques) de back-end

6.5 Challenges rencontrés et apprentissage

6.5.1 Challenges rencontrés et solutions appliquées

Les deux challenges principaux de cette version ont été les suivants:

1. Challenges techniques

   Bien que pas très exigeante techniquement, cette version pose les fonctionnalités de base sur lesquelles les versions suivantes reposent. En ce qui concerne les algorithmes, le challenge principal a été la recherche de la documentation qui prend du temps. Mais bien que cela ait demandé du temps, nous avons eu la chance de trouver beaucoup de documentation de bonne qualité sur le sujet des tatamis. Il nous a ensuite été relativement facile d'implémenter et tester les solutions trouvées.

2. Challenges organisationnels

   Bien que la coopération au sein de l'équipe ait débuté en phase de pré-développement, ce sprint était le premier qui impliquait un travail réel sur le code qui pose forcément

[2] Filling rectangular rooms with Tatami mats, 2002

de nouveaux challenges. Nous avons par ailleurs perdu définitivement un membre de l'équipe, impliquant inévitablement plus de travail de la part des membres restants. Pour nous permettre de bien avancer, nous avons mis en application les principes organisationnels suivants:

- Sessions de travail régulières (hebdomadaires) pour discuter des points ouverts et répartir les tâches. Fréquence plutôt élevée pour garder un bon rythme de travail et éviter les à-coups.
- Retranscription écrite claire des tâches à effectuer avec les dates butoirs (et dépendances de tâches lors qu'il y en a) avec un Compte Rendu écrit pour chaque session de travail
- Communication entre les sessions de travail (par Slack), notamment pour les discussions sur l'exécution des tâches pré-requises à d'autres tâches dépendantes.
- Travail personnel entre les session pour accomplir les tâches

Par ailleurs, pour gérer le code, github (intégré à Slack pour recevoir les notifications lors des updates) a été utilisé.

6.5.2 Apprentissage

Les enseignements principaux de ce sprint sont les suivants:

1. Le choix de méthodologie agile confirmé comme étant la bonne approche pour travailler sur le projet.
2. La perte d'un membre de l'équipe aurait notamment été plus difficile à gérer si tout avait été planifié de manière rigide au départ
3. Le bon fonctionnement de la coopération et organisation choisie (régularité des sessions de travail, documentation des sessions de travail avec des tâches et dates butoirs claires,...)

7. Version Beta

7.1 Backlog/roadmap du sprint Beta

L'objectif de ce sprint est de donner bien plus de valeur à l'utilisateur par:

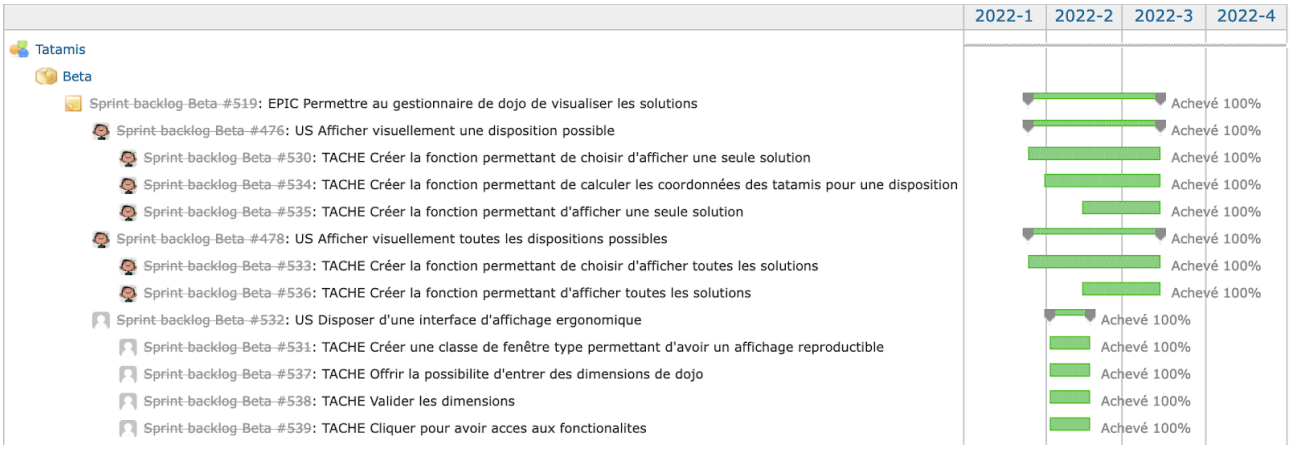
1. Une visualisation des dispositions possible (ce qui apporte bien plus à l'utilisateur que la version Alpha)
2. Une utilisation facilité par une interface utilisateur intuitive et ergonomique.

Pour ce faire le backlog suivant a été "embarqué" dans cette version et a résulté en la roadmap suivante:

#	Sujet	En tant que...	je veux...	afin de...	Type	Tâche parente	Assigné à	Début	Echéance	% réalisé
519	EPIC Permettre au gestionnaire de dojo de visualiser les solutions	En tant que gestionnaire de dojo,	je souhaite visualiser une ou plusieurs dispositions possibles,	afin de m'aider à placer les tatamis sur mon dojo	Epic			25/01/2022	14/03/2022	
539	TACHE Cliquer pour avoir acces aux fonctionalites	Gestionnaire de dojo	pouvoir simplement cliquer sur un bouton pour acceder a une fonctionalite	faciliter l'utilisation	Tache	~~Sprint backlog Beta #532~~	Benoît Anger	02/02/2022	16/02/2022	
538	TACHE Valider les dimensions	Gestionnaire de dojo	pouvoir entrer uniquement des dimensions valides (entiers positifs et inferieur a la limite de performance du programme)	faciliter l'utilisation	Tache	~~Sprint backlog Beta #532~~	Benoît Anger	02/02/2022	16/02/2022	
537	TACHE Offrir la possibilite d'entrer des dimensions de dojo	Gestionnaire de dojo	entrer les dimensions de mon dojo	pouvoir acceder aux fonctionalites	Tache	~~Sprint backlog Beta #532~~	Benoît Anger	02/02/2022	16/02/2022	
536	TACHE Créer la fonction permettant d'afficher toutes les solutions	équipe de développement	implémenter une fonction	afficher dans l'interface utilisateur toutes les solutions	Tache	~~Sprint backlog Beta #478~~	Bruno Bourgine	14/02/2022	14/03/2022	
535	TACHE Créer la fonction permettant d'afficher une seule solution	équipe de développement	implémenter une fonction	afficher dans l'interface utilisateur une seule solution	Tache	~~Sprint backlog Beta #476~~	Bruno Bourgine	14/02/2022	14/03/2022	
534	TACHE Créer la fonction permettant de calculer les coordonnées des tatamis pour une disposition	équipe de développement	créer une fonction permettant d'obtenir les coordonnées des tatamis pour une disposition	tracer les tatamis dans la fenêtre d'affichage	Tache	~~Sprint backlog Beta #476~~	Bruno Bourgine	31/01/2022	14/03/2022	

533	TACHE Créer la fonction permettant de choisir d'afficher toutes les solutions	équipe de développement	implémenter une fonction	donner la possibilité à l'utilisateur de choisir d'afficher toutes les solutions	Tache	~~Sprint backlog Beta #478~~	Bruno Bourgine	25/01/2022	14/03/2022	
531	TACHE Créer une classe de fenêtre type permettant d'avoir un affichage reproductible	équipe de développement	implémenter une classe de fenêtre type	pouvoir l'instanciée à chaque demande d'affichage	Tache	~~Sprint backlog Beta #532~~	Benoît Anger	02/02/2022	16/02/2022	
530	TACHE Créer la fonction permettant de choisir d'afficher une seule solution	équipe de développement	implémenter une fonction	donner la possibilité à l'utilisateur de choisir d'afficher une seule solution	Tache	~~Sprint backlog Beta #476~~	Bruno Bourgine	25/01/2022	14/03/2022	
532	US Disposer d'une interface d'affichage ergonomique	Gestionnaire de dojo	disposer d'une interface d'affichage ergonomique	me faciliter la visualisation des dispositions	User story	~~Sprint backlog Beta #519~~	Benoît Anger	02/02/2022	16/02/2022	
478	US Afficher visuellement toutes les dispositions possibles	En tant que gestionnaire de dojo,	je souhaite visualiser l'ensemble des dispositions possibles,	afin de m'aider à placer les tatamis sur mon dojo	User story	~~Sprint backlog Beta #519~~	Bruno Bourgine	25/01/2022	14/03/2022	
476	US Afficher visuellement une disposition possible	En tant que gestionnaire de dojo,	je souhaite visualiser une disposition possible,	afin de m'aider à placer les tatamis sur mon dojo	User story	~~Sprint backlog Beta #519~~	Bruno Bourgine	25/01/2022	14/03/2022	

Et ici présenté en diagramme de Gantt:



7.2 Tests

Les tests réalisés pour cette version et leurs résultats sont les suivants:

id	Sujet	Test d'acceptance	Méthode de test	Résultat
539	TACHE Cliquer pour avoir accès aux fonctionnalités	Quand l'application est lancée, les boutons s'affichent	Manuel	OK
		Quand l'utilisateur clique sur un bouton, la fonctionnalité est activée.	Manuel	OK
538	TACHE Valider les dimensions	Quand l'application est lancée, seules des valeurs entières peuvent être entrées	Manuel	OK
		Quand l'application est lancée, les valeurs entrables sont restreintes aux valeurs dans un intervalle défini.	Manuel	OK
		Quand l'utilisateur omet ou entre 0 pour au moins une dimension, un message d'erreur s'affiche.	Manuel	OK

id	Sujet	Test d'acceptance	Méthode de test	Résultat
537	TACHE Offrir la possibilité d'entrer des dimensions de dojo	Quand l'application est lancée, une fenêtre s'ouvre avec la possibilité d'entrer les dimensions.	Manuel	OK
536	TACHE Créer la fonction permettant d'afficher toutes les solutions	Quand la fonction reçoit en input les coordonnées des tatamis pour une disposition dimensions, alors elle retourne un graph avec toutes les solutions possibles"	Manuel	OK
535	TACHE Créer la fonction permettant d'afficher une seule solution	Quand la fonction reçoit en input les coordonnées des tatamis pour une disposition dimensions, alors elle retourne un graph avec une solution possible"	Manuel	OK
534	TACHE Créer la fonction permettant de calculer les coordonnées des tatamis pour une disposition	Quand la fonction reçoit en input les dimensions d'un dojo, alors elle retourne les coordonnées des tatamis pour une disposition"	Manuel	OK
533	TACHE Créer la fonction permettant de choisir d'afficher toutes les solutions	Étant donné les inputs des dimensions d'un dojo, quand cette fonction est choisie, elle retourne un graph avec toutes les solutions possibles"	Manuel	OK
532	US Disposer d'une interface d'affichage ergonomique	Quand le programme est lancé, il ouvre une interface ergonomique"	Manuel	OK
531	TACHE Créer une classe de fenêtre type permettant d'avoir un affichage reproductible	Quand l'application est lancée, une fenêtre s'ouvre avec les bonnes (adaptées à l'écran) et memes dimensions"	Manuel	OK
530	TACHE Créer la fonction permettant de choisir d'afficher une seule solution	Étant donné les inputs des dimensions d'un dojo, quand cette fonction est choisie, elle retourne un graph avec une seule solution possible"	Manuel	OK
519	EPIC Permettre au gestionnaire de dojo de visualiser les solutions	cf. tests utilisateurs des US		OK
		Le nombre de solution du programme donne le même nombre de solution que le calcul de coordonnées Tatamis	Automatisé	OK
478	US Afficher visuellement toutes les dispositions possibles	Étant donné des dimensions d'un dojo saisies, quand il sélectionne cette option, il obtient visuellement toutes les dispositions possibles"	Manuel	OK
476	US Afficher visuellement une disposition possible	Étant donné des dimensions d'un dojo saisies, quand il sélectionne cette option, il obtient visuellement une disposition possible"	Manuel	OK

Les tests automatisés sont définis dans le fichier `testbeta.py`. Ils peuvent être reproduits en l'exécutant avec ligne de commande `python3 -m pytest`.

7.3 Documentation utilisateur Beta

7.3.1 Prérequis

Configuration et installations requises:

- Python 3.9 ou supérieur
- Librairies Python: datetime, numpy, PyQt5.QtCore, PyQt5.QtGui, PyQt5.QtWidgets, sys, time.

7.3.2 Comment trouver le nombre de tatamis nécessaires pour remplir un dojo?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet.

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: "Connaître le nombre de tatamis 2x1 nécessaires pour la taille du dojo" *Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.*

4. Interpréter la réponse:

   (a) **Réponse**: "Le nombre de tatamis 2x1 nécessaires pour ce dojo est : [Nombre]". **Interprétation**: il existe au moins une disposition possible et vous aurez besoin d'exactement [Nombre] tatamis pour remplir le dojo.

   (b) **Réponse**: "Le nombre de tatamis 2x1 nécessaires pour ce dojo est : Aucune disposition possible de tatamis 2x1 pour ce dojo". **Interprétation**: il n'existe aucune disposition possible de tatamis 2 x 1 pour remplir le dojo.

7.3.3 Comment savoir s'il existe une disposition possible de tatamis pour un dojo donné?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet.

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: "Savoir s'il existe une disposition pour le dojo"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.

4. Interpréter la réponse:

   (a) **Réponse**: "Il existe au moins une disposition avec des tatamis 2x1 pour ce dojo". **Interprétation**: il existe au moins une disposition possible.

   (b) **Réponse**: "Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo". **Interprétation**: il n'existe aucune disposition possible de tatamis 2 x 1 pour remplir le dojo. Il est dans ce cas probable de devoir utiliser des demi-tatamis pour remplir pleinement le dojo.

7.3.4 Comment savoir combien il existe des dispositions possibles de tatamis pour un dojo donné?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet.

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: "Connaître le nombre dispositions possibles".

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.

4. Interpréter la réponse:

   (a) **Réponse**: "Il existe [Nombre] dispositions possibles". **Interprétation** : il existe des dispositions pour ce dojo et [Nombre] est le nombre de dispositions possibles pour remplir le dojo.

   (b) **Réponse**: "Il existe 0 disposition possible". **Interprétation**: la demande est non pertinente car il n'existe pas de disposition possible de tatamis 2x1 pour les dimensions du dojo.

7.3.5 Comment afficher une disposition?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet.

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: "Afficher une disposition"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.

4. Une disposition s'affiche

   Ou bien le message d'erreur suivant s'affiche: "Demande impossible. Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo" apparaît, ce qui signifie que la demande est non pertinente car il n'existe pas de disposition possible de tatamis 2x1 pour les dimensions du dojo.

7.3.6 Comment afficher toutes les dispositions possibles?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)
2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet.

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*
3. Cliquer sur le bouton: "Afficher toutes les dispositions possibles"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.
4. Toutes les dispositions possibles s'affichent

   Ou bien le message d'erreur suivant s'affiche: "Demande impossible. Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo" apparaît, ce qui signifie que la demande est non pertinente car il n'existe pas de disposition possible de tatamis 2x1 pour les dimensions du dojo.

7.4 Explication des algorithmes et choix de programmation

7.4.1 Algorithme pour l'affichage des dispositions

Première solution envisagée

La première solution envisagée afin d'afficher la disposition des tatamis comprenait l'utilisation de la bibliothèque python facile, dont une application au pavage de surface par tatamis a été trouvée sur le site de Xavier Olive[1]. La publication semblait correspondre exactement à ce que nous souhaitions, à savoir un calcul des dispositions possibles, et un affichage graphique adapté (via la bibliothèque matplotlib).

La bibliothèque facile permet de modéliser et résoudre un problème en programmant ses contraintes. Les positions et les directions des tatamis constituent les variables du problème.

Les contraintes étant constituées par les assertions suivantes:

- deux tatamis ne peuvent pas se chevaucher
- les tatamis ne sortent pas du cadre
- quatre tatamis ne peuvent pas se rejoindre en un point

L'utilisation des algorithmes proposés comportait néanmoins plusieurs inconvénients :

- Un temps de calcul devenant rapidement bloquant pour des dimensions de dojo encore raisonnables (16×16).

[1] https://www.xoolive.org/2016/02/29/pavage-par-tatamis.html

- Une bibliothèque complexe à appréhender, elle-même issue d'une adaptation en python de fonctionnalités développées en OCaml.

Nous aurions pu nous contenter de cette solution, mais l'impossibilité d'obtenir des pavages pour des dimensions au-delà de 16×16 nous a semblé rédhibitoire.

Solution retenue

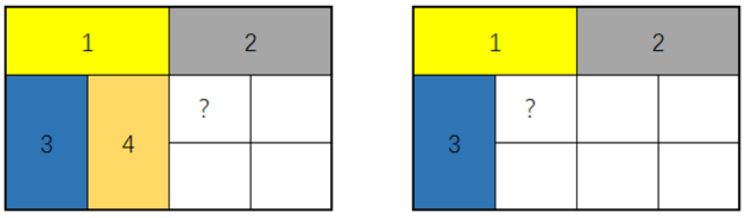
Après une recherche un peu plus poussée, il s'est avéré que ce problème de pavage dit "tatamis-parfait" fait l'objet d'une question dans un livre édité en chinois et dont la traduction anglaise proposée pour le titre est : "Programmer's algorithm interesting topic". Les problèmes traités dans ce livre le sont initialement en Ruby et Javascript, néanmoins un internaute chinois propose une interprétation de l'algorithme en python sur une note de blog[2]. Celle-ci ayant été traduite en anglais[3], nous avons pu en prendre plus facilement connaissance et ainsi en exploiter les idées.

L'auteur de cette note propose de modéliser la pièce en la découpant selon une grille d'intervalle 1.

Le problème est identifié comme celui d'un parcours de graphe (arbre binaire) en profondeur d'abord. La racine de l'arbre étant constituée par une pièce vide de tatami. Chaque nœud du graphe correspond à un état de pavage donné, où chaque zone de la grille contient un entier correspondant au numéro du tatami qui la recouvre. Le nombre 0 désigne alors une zone vide de tatami.

La pièce est donc modélisée par une matrice de dimension H*W où H désigne sa hauteur et W sa largeur. Afin de détecter les bords de la pièce lors du parcours, la matrice est "bordée" par le nombre -1.

Le cœur du problème consiste à savoir si une zone de la grille peut être recouverte par un tatami. Étant donné que nous ne voulons pas que quatre tatamis se rejoignent en même coin, le critère est donc qu'une zone ne peut être pavée que si les trois positions de la grille situées à gauche, en haut à gauche, et en haut ne sont pas toutes pavées par des tatamis différents. Cette situation est traduite par l'illustration ci-dessous :



L'algorithme parcourt la grille ligne par ligne et colonne par colonne en vérifiant pour chaque position si un tatamis peut-être posé : dans un premier temps en position horizontale, puis dans un deuxième temps en position verticale. Si c'est le cas, la position de la grille

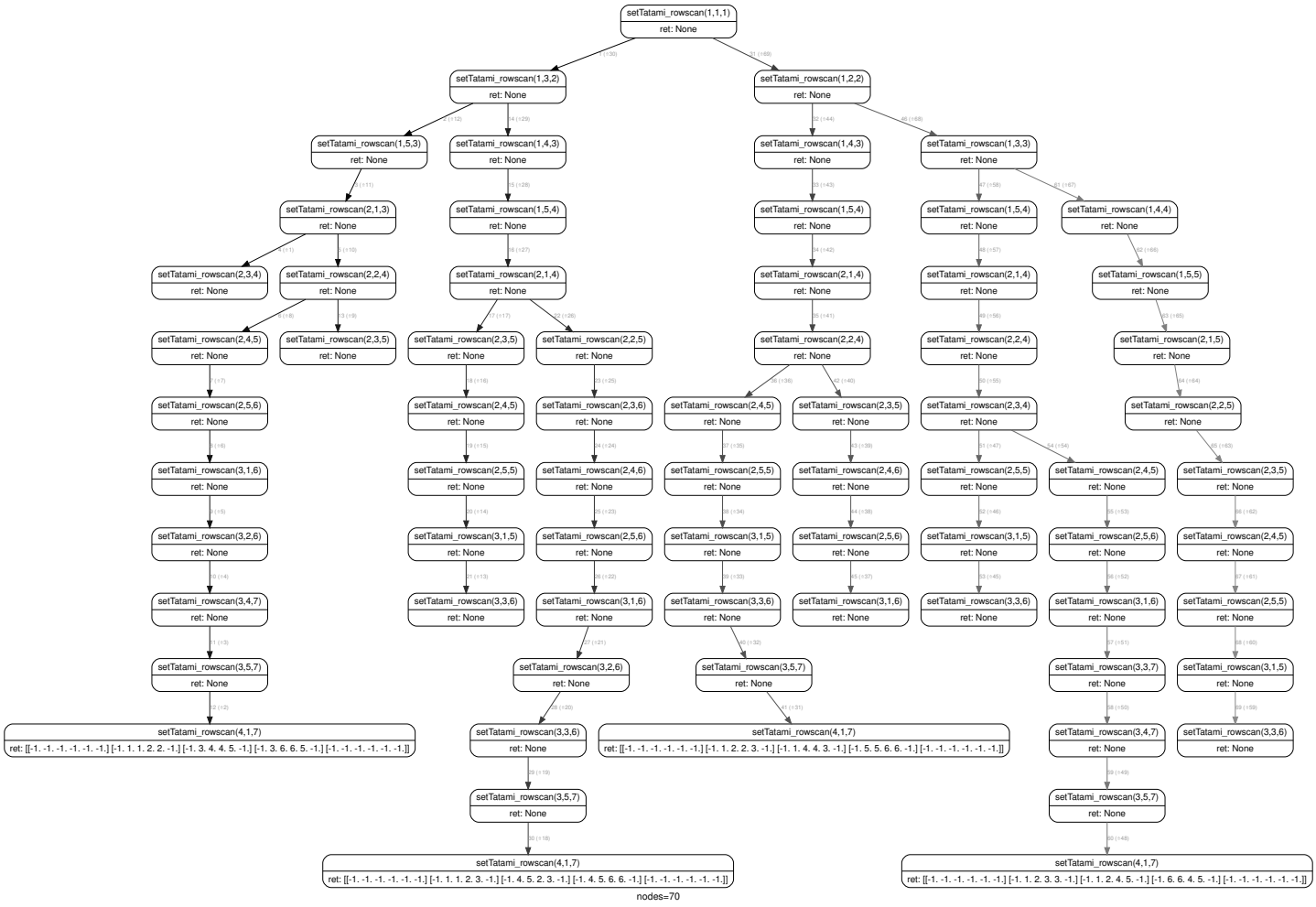
[2]https://blog.csdn.net/chenxy_bwave/article/details/120364982

[3]https://www.fatalerrors.org/a/programmer-s-algorithm-interesting-topic-q32-laying-method-of-tatami.html

ainsi que sa zone adjacente (horizontale ou verticale) reçoivent le numéro du tatami courant et l'algorithme est appelé pour la position suivante, en incrémentant le numéro du tatami. Lorsque la ligne atteinte correspond au bord bas du tatamis, le pavage est complet.

L'auteur précise qu'un tel parcours peut comporter une profondeur considérable selon les dimensions de la pièce à paver et que selon les cas on peut donc atteindre rapidement la profondeur de récursivité maximale.

L'illustration d'un parcours de recherche pour un pavage (4×3) est donnée ci-dessous.



En testant le programme en Python proposé par l'auteur, nous atteignons effectivement une limite de calcul, mais celle-ci nous permet néanmoins de traiter des dojos plus grands que la solution évoquée précédemment. Nous augmentons ainsi notre capacité de traitement d'environ 80% puisque la dimension limite de dojo passe de 16 à 25.

La publication originale fournissait l'implémentation d'une fonction récursive manipulant des variables globales. Afin de pouvoir utiliser cette solution dans notre projet, la fonction a été intégrée au sein d'une classe en tant que méthode, et les variables globales sont devenues des attributs de celle-ci. La fonction initiale ne retournant qu'une matrice d'index, il a fallu ajouter une méthode de classe afin de produire à partir de cette matrice,

les positions de chaque tatami dans le plan. L'enregistrement des positions se faisant à l'aide d'une liste de dictionnaire, chacun de ces dictionnaires contenant les caractéristiques d'un tatami (index, largeur, longueur, position verticale et position horizontale).

7.4.2 Choix de programmation Interface

Les choix importants concernant l'interface ont été les suivants:

Choix de la librairie d'interface graphique

Une recherche initiale a permis d'identifier 5 librairies à notre disposition:

- PyQt5
- Tkinter
- Pyside2
- Kivy
- wxPython

Trois critères de choix ont été appliqués:

- Capacités de la librairie
- Disponibilité de la documentation et ressources en ligne
- Connaissances préalables par l'équipe et simplicité

L'évaluation a conclu aux résultats suivants:

		Tkinter	Pyside2	Kivy	wxPython
Capacités de la librairie	Très large variété de widgets (boutons, champs de saisie, boîtes de message...) Beaucoup de possibilités Grille disponible	Large variété de widgets (boutons, champs de saisie, boîtes de message...) Grille disponible (mais également une fonctionnalité qui facilite le placement)	Large variété de widgets (boutons, champs de saisie, boîtes de message...) Grille disponible	Widgets disponibles, dont certains bien designés mais d'autres moins intuitifs (par ex. les boutons) Grille disponible	Large variété de widgets (boutons, champs de saisie, boîtes de message...) Grille disponible
Disponibilité de la documentation et ressources en ligne	20 000 questions sur Stackoverflow	45 000 questions sur Stackoverflow	1 500 questions sur Stackoverflow	12 000 questions sur Stackoverflow	7 000 questions sur Stackoverflow
Connaissances préalables et simplicité	Connaissances préalables de l'équipe	Peu de connaissances Mais facile a comprendre	Non	Non	Non

Compte tenu de cette évaluation, le choix entre PyQt5 et Tkinter n'a pas été évident. Étant donné les connaissances préalables de l'équipe en PyQt5 et les possibilités plus importantes de la librairie par rapport à Tkinter, c'est PyQt5 qui a finalement été choisi.

Choix concernant la disposition des éléments de l'interface

Deux options ont été étudiées:

1. Positionnement spécifique de chaque élément de l'interface (par la fonction `move()` de PyQt5).
    - *Avantage* : Flexibilité (possibilité de placer très précisément chaque objet sur les axes abscisse et ordonnées de la fenêtre).
    - *Inconvénient* : Fastidieux (nécessite de placer chaque objet avec ses coordonnées et les modifications de disposition - notamment par l'ajout d'objet - s'en trouve très longs à exécuter).
2. Positionnement par la mise en place et utilisation d'une grille (invisible à l'utilisateur) pour placer les éléments (par la sous-librairie QGridLayout de PyQt5).
    - *Avantage* : Rapidité à coder (possibilité de placer très précisément chaque objet sur les axes des abscisses et des ordonnées de la fenêtre).

      Bon alignement obtenu très facilement obtenu (car la grille assure le bon alignement).
    - *Inconvénient* : Moins de flexibilité pour placer précisément les objets.

Étant donné les avantages importants de l'option 2 et le fait que l'interface pour notre programme pouvait facilement être mise en place sous forme d'une grille, l'option 2 a été choisie.

Choix concernant l'intuitivité de l'interface

Pour cette première 'vraie' interface pour l'utilisateur (sachant qu'en version Alpha, l'utilisateur interagissait avec le programme par l'intermédiaire du Terminal), des choix importants et engageants pour la suite ont été à faire. De manière générale, il a été choisi de mettre un focus particulier sur la facilité d'utilisation et l'intuitivité pour l'utilisateur. Pour ce faire voilà les principales questions qui ont été étudiées et choix effectués:

1. Appel des fonctionnalités

   Pour une utilisation facile et compréhension des fonctionnalité disponible, il a été choisi d'utiliser des boutons pour l'appel des fonctionnalités, et de laisser beaucoup d'espace dans le boutons pour y afficher les fonctionnalité en détails

2. Manière d'exprimer les retours

   Pour une qualité d'utilisation, tous les retours (message d'erreur ou réponse attendue par l'utilisateur) sont exprimés par des fenêtres "pop up"

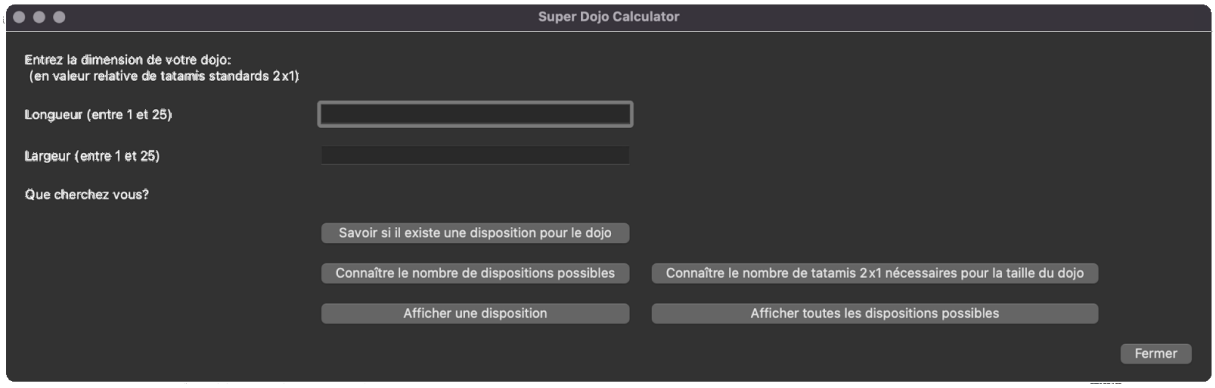
3. Guidance d'utilisation, "formation" des utilisateurs et information

   Il a été choisi de mettre un focus particulier sur ces aspects pour fournir une grande qualité de programme. L'objectif est de guider au maximum l'utilisateur (pour "borner" ses actions à ce qui est autorisé) et de fournir des messages de retour clairs pour chaque type d'erreur ou d'impossibilité d'utilisation de fonctionnalités. Voici les principales mesures mises en place :

- Les champs de saisie sont extrêmement guidés par une validation (à l'aide de la fonction `QIntValidator`): l'utilisateur ne peut saisir que des entiers entre les valeurs inscrites sur l'interface
- Les seules saisies impossibles à contrôler sont la saisie de "0" ou l'absence de saisie, mais des messages d'erreur ont été mis en place.
- Des tests sont effectués avant l'appel de chaque fonctionnalité pour savoir si la fonctionnalité est disponible avec les dimensions de dojo saisies. Dans le cas contraire, un message d'erreur clair est donné à l'utilisateur.
- Tous les cas d'erreurs ont été traités (un utilisateur ne peut pas se retrouver bloqué en produisant une erreur qui ne retourne pas une explication sur comment la résoudre)

Interface et messages de retour:

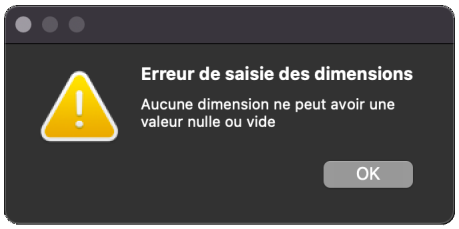
- Interface :



- Exemple de message de retour (succès):

- Exemple de message indiquant à l'utilisateur que la demande est impossible avec les dimensions saisies:

- Exemple de message d'erreur en cas de non saisie ou saisie nulle de dimensions:



7.4.3 Choix de la structure du programme

Dans la continuité des choix fait en version Alpha, nous avons mis en application des principes *SOLID*, et en particulier du *Single Responsibility Principle* et *Open-Closed Principle*:

- Un fichier a une fonction principale Dans cette version beta, nous sommes passés de 2 fichiers (un fichier de front-end `interface.py` et un fichier de back-end `alpha.py`) à 4 fichiers du fait de la croissance des fonctionnalités et des besoins de calculs en back-end. Nous avons donc:
  - `interface.py`: fichier qui continue à contenir toutes les fonctionnalités de front-end, c'est à dire l'interface utilisateur
  - `calcul_coordonnees_tatamis.py`: classe qui calcule les coordonnées des tatamis d'après les dimensions du dojo. Cette classe contient la fonction permettant de calculer une solution de pavage, d'après le code trouvé sur la publication citée dans la partie 7.4.1, à laquelle a été adjointe une fonction permettant d'obtenir les coordonnées des tatamis dans le plan.
  - `calcul_nombre_dispositions.py`: fichier qui reprend les fonctions de calculs (basiques) de back-end de la version alpha.
  - `dojo.py` : classe permettant d'instancier un tatamis d'après ces propriétés: position, dimensions, couleur. L'affichage des solutions graphiques se faisant dans une fenêtre à part et utilisant des fonctions particulières, cela justifiait l'existence d'une classe spécifique.
- Chaque fonction a un seul usage. Plusieurs exemples peuvent être cités dans le fichier `interface.py`: de nombreuses fonctions à fonctionnalité très limitée mais réutilisables ont été créées comme par exemple `set_longueur()`, `set_largeur()`, `valeur_vide(self)`, `clickExiste()` (leur fonctionnalité est documentée dans le code et donc non-reprise dans ce rapport)
- Dans la mesure du possible des fonctions ou classes génériques sont créées puis réutilisées. Un bon exemple a cela sont les classes génériques de l'interface: `MessageSaisieInvalide`, `MessageDemandeImpossible` et `MessageInfo`.

7.5 Challenges rencontrés et apprentissage

7.5.1 Challenges rencontrés et solutions appliquées

Les deux challenges principaux de cette version ont été les suivants:

1. Challenges techniques

   Comme on peut le constater plus dans la section sur les choix de programmation, il s'agit de la version la plus complexe techniquement. D'un point de vue de l'algorithme, l'affichage des dispositions a été un gros challenge technique. D'un point de vue de l'interface, beaucoup de choix structurants ont été nécessaires, avec des essais et révisions. Enfin, d'un point de vue de l'architecture du programme, les choix faits en version Alpha ont été à implémenter de manière plus poussée. Pour faire face à ces challenges, nous avons adopté l'approche suivante:

   - Planification en systématiquement décomposant les "gros" problèmes en problèmes plus petits
   - Recherches techniques avant tout choix à faire
   - Revue des choix avant implémentation

2. Challenges organisationnels et charge de travail

   La charge de travail de cette version a été sous-estimée au moment de la planification et de la décision des "User story" à embarquer dans cette version. En effet, la lourdeur de cette version, combinée aux choix techniques importants à faire, et à une équipe de taille réduite (2 personnes) a été un challenge important.

   L'organisation de l'équipe et du travail est ce qui nous a permis de faire face à ce problème et de respecter le calendrier prévu. Les clés de l'organisation sont restées:

   - Sessions de travail pour discuter des points ouverts et repartir les taches
   - Retranscription écrite claire des tâches à effectuer avec les dates butoir
   - Communication entre les sessions de travail (par Slack)
   - Travail personnel entre les session pour accomplir les tâches

7.5.2 Apprentissage

Les deux principaux apprentissages sont les suivants:

1. Importance des recherches

   Les challenges techniques importants à surmonter nous ont permis de confirmer l'importance de faire des recherches et planifier la solution avant d'implémenter. Cela évite de perdre du temps à essayer des implémentations mal pensées. La réussite de cette version nous apprend également qu'avec des recherches, des problèmes qui semblent difficiles peuvent être surmontés.

2. Planifier du sprint et de la charge de travail

   Si cela était à refaire, nous aurions sûrement été moins ambitieux pour la version Beta en décalant des User Stories à la version Release Candidate. Le travail sur cette version nous fait réaliser à la fois l'importance de bien "quantifier" le volume de travail des User Stories, mais en même temps la difficulté à le faire. En effet, en particulier en ce début de projet, il reste difficile d'estimer avec précision la charge de travail nécessaire à l'implémentation des User Stories.

3. Promotion en classe

   La version alpha comporte essentiellement des algorithmes écrits sous forme de fonctions. Du fait de la complexification des challenges et de l'ajout d'une interface graphique, la programmation objet s'est imposée à nous. Tout d'abord de part l'utilisation de la bibliothèque PyQt et des fonctionnalités qu'il a fallu développées en s'appuyant sur celle-ci, et ensuite de part les caractéristiques des solutions nécessitant d'être encapsulées et de disposer d'attribut et de fonctionnalités propres.

8. Version Release Candidate

8.1 Backlog/roadmap du sprint Release Candidate

L'objectif de ce sprint est de donner bien plus de valeur à l'utilisateur par l'ajout de nombreuses fonctionnalités supplémentaires. Un objectif secondaire a été l'amélioration de l'interface d'un point de vue visuel et pour en faciliter la compréhension.

Pour ce faire le backlog suivant a été "embarqué" dans cette version et a résulté en la roadmap suivante:

8.2 Tests

id	Sujet	Test d'acceptance	Méthode de test	Résultat
540	EPIC Amélioration de l'expérience par ajout de fonctionnalités et amélioration de l'interface	cf tests utilisateur des US	Manuel	OK
481	US Affichage des dimensions sur les représentation graphiques des dispositions de tatamis	L'affichage graphique des solutions comprend un rappel des dimensions du dojo.	Manuel	OK
482	US Modification des dimensions du dojo	L'application permet de saisir d'autres dimensions que celles initialement renseignées.	Manuel	OK
482	US Modification des dimensions du dojo	L'application permet de saisir d'autres dimensions que celles initialement renseignées.	Manuel	OK
484	US Comprendre s'il est utile d'utiliser des demi-tatamis pour trouver une solution	Si il n'existe pas de solutions avec des tatamis entiers, l'application affiche qu'il est toujours possible d'obtenir une solutions avec des demi-tatamis.	Manuel	OK
525	US Obtenir une solution étant donné un nombre de tatamis	Lorsque l'utilisateur entre un nombre de tatamis, l'application propose une solution dont une des dimensions ne peut-être inférieure à 3 et dont le rapport largeur/longueur ne peut être inférieur à 1/3.	Manuel	OK
551	TACHE créer une fonction pour obtenir une solution étant donné un nombre de tatamis	Lorsque l'utilisateur entre un nombre de tatamis, l'application propose une solution dont une des dimensions ne peut-être inférieure à 3 et dont le rapport largeur/longueur ne peut être inférieur à 1/3.	Manuel	OK
552	TACHE permettre à l'utilisateur d'utiliser la fonction pour obtenir une solution étant donné un nombre de tatamis	La fonctionnalité est disponible sur l'interface utilisateur	Manuel	OK
527	US calculer les dispositions pour des dimensions plus petites	Lorsque l'utilisateur propose des dimensions, il n'existe pas de disposition aux dimensions plus grandes entre la(les) solution(s) alternative(s) proposée(s) par l'application et la demande de l'utilisateur.	Manuel	OK
544	TACHE créer une fonction pour calculer les dispositions pour des dimensions plus petites	Lorsque l'utilisateur propose des dimensions, il n'existe pas de disposition aux dimensions plus grandes entre la(les) solution(s) alternative(s) proposée(s) par l'application et la demande de l'utilisateur.	Automatisé	OK
545	TACHE permettre à l'utilisateur d'utiliser la fonction de calcul des dispositions pour des dimensions plus petites	La fonctionnalité est disponible sur l'interface utilisateur	Manuel	OK

id	Sujet	Test d'acceptance	Méthode de test	Résultat
541	US Amélioration de l'expérience par ajout de fonctionnalités et amélioration de l'interface	Chaque interaction proposée par l'interface correspond à une fonctionnalité bien définie et identifiable, l'interface est intuitive est agréable visuellement.	Manuel	OK
547	TACHE Interface: changement de couleur de fonds	Test utilisateur: Le fonds de l'interface est coloré	Manuel	OK
548	TACHE Interface: amélioration des champs de saisie	Les champs de saisie sont bien identifiables	Manuel	OK
549	TACHE Interface: changement des couleurs et police du texte	Le texte est coloré et avec une police lisible et moderne	Manuel	OK
550	TACHE Interface: redisposition et catégorisation des fonctionnalités	Les fonctionnalités sont groupées par theme sur l'interface"	Manuel	OK
542	US Afficher la surface	La valeur d'aire affichée sur l'interface correspond à l'aire de la surface occupée par les tatamis.	Manuel	OK

8.3 Documentation utilisateur Release Candidate

8.3.1 Prérequis

Configuration et installations requises:

- Python 3.9 ou supérieur
- Librairies Python: datetime, numpy, PyQt5.QtCore, PyQt5.QtGui, PyQt5.QtWidgets, sys, time.
- Installer les polices Nexa (light et bold) (fichiers otf fournis)

8.3.2 Comment trouver la surface du dojo?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)
2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l'interface intitulés "Longueur (entre 1 et 25)" et "Largeur (entre 1 et 25)").

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*
3. Cliquer sur le bouton: "Afficher la surface du dojo" *Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.*
4. Interpréter la réponse:

   (a) **Réponse**: "La surface du dojo est [Nombre]". **Interprétation**: la surface du dojo est de [Nombre] (unité identique à celle des dimensions entrées).

8.3.3 Comment trouver le nombre de tatamis nécessaires pour remplir un dojo?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)
2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l'interface intitulés "Longueur (entre 1 et 25)" et "Largeur (entre 1 et 25)").

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*
3. Cliquer sur le bouton: "Connaître le nombre de tatamis 2×1 nécessaires pour la taille du dojo" *Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.*

4. Interpréter la réponse:

   (a) **Réponse**: "Le nombre de tatamis 2x1 nécessaires pour ce dojo est : [Nombre]". **Interprétation**: il existe au moins une disposition possible et vous aurez besoin d'exactement [Nombre] tatamis pour remplir le dojo.

   (b) **Réponse**: "Demande impossible. Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo". **Interprétation**: il n'existe aucune disposition possible de tatamis 2 x 1 pour remplir le dojo.

8.3.4 Comment savoir s'il existe une disposition possible de tatamis pour un dojo donné?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l'interface intitulés "Longueur (entre 1 et 25)" et "Largeur (entre 1 et 25)"). *Nb:*

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: "Savoir s'il existe une disposition pour le dojo"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.

4. Interpréter la réponse:

   (a) **Réponse**: "Il existe au moins une disposition avec des tatamis 2x1 pour ce dojo". **Interprétation**: il existe au moins une disposition possible.

   (b) **Réponse**: "Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo". **Interprétation**: il n'existe aucune disposition possible de tatamis 2 x 1 pour remplir le dojo. Il est dans ce cas probable de devoir utiliser des demi-tatamis pour remplir pleinement le dojo.

8.3.5 Comment savoir combien il existe de dispositions possibles de tatamis pour un dojo donné?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l'interface intitulés "Longueur (entre 1 et 25)" et "Largeur (entre 1 et 25)"). *Nb:*

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: “Connaître le nombre dispositions possibles”.

   Nb: Si vous oubliez d’entrer une dimension ou si vous entrez une dimension 0, un message d’erreur “Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide” apparaît.

4. Interpréter la réponse:

   (a) **Réponse**: “Il existe [Nombre] dispositions possibles”. **Interprétation** : il existe des dispositions pour ce dojo et [Nombre] est le nombre de dispositions possibles pour remplir le dojo.

   (b) **Réponse**: “Il existe 0 disposition possible”. **Interprétation**: la demande est non pertinente car il n’existe pas de disposition possible de tatamis 2x1 pour les dimensions du dojo.

8.3.6 Comment afficher une disposition?

1. Lancer l’interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l’interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l’interface intitulés “Longueur (entre 1 et 25)” et “Largeur (entre 1 et 25)”).

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n’a pas d’importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: “Afficher une disposition”

   Nb: Si vous oubliez d’entrer une dimension ou si vous entrez une dimension 0, un message d’erreur “Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide” apparaît.

4. Une disposition s’affiche ainsi que le rappel des dimensions du dojo, sa surface et le nombre de tatamis nécessaires pour le remplir.

   Ou bien le message d’erreur suivant s’affiche: “Demande impossible. Il n’existe pas de disposition possible avec des tatamis 2x1 pour ce dojo” apparaît, ce qui signifie que la demande est non pertinente car il n’existe pas de disposition possible de tatamis 2x1 pour les dimensions du dojo.

8.3.7 Comment afficher toutes les dispositions possibles?

1. Lancer l’interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l’interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l’interface intitulés “Longueur (entre 1 et 25)” et “Largeur (entre 1 et 25)”).

   Nb:

- *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
- *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: "Afficher toutes les dispositions possibles"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.

4. Toutes les dispositions possibles s'affichent ainsi que le rappel des dimensions du dojo, sa surface et le nombre de tatamis nécessaires pour le remplir.

   Ou bien le message d'erreur suivant s'affiche: "Demande impossible. Il n'existe pas de disposition possible avec des tatamis 2x1 pour ce dojo" apparaît, ce qui signifie que la demande est non pertinente car il n'existe pas de disposition possible de tatamis 2x1 pour les dimensions du dojo.

8.3.8 En cas d'absence de solutions, comment connaître la taille maximum du dojo qui permettrait de trouver au moins un disposition?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)

2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l'interface intitulés "Longueur (entre 1 et 25)" et "Largeur (entre 1 et 25)").

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*

3. Cliquer sur le bouton: "Quelle taille maximum de dojo pour avoir une solution"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.

4. Interpréter la réponse :

   (a) **Réponse**: "Taille maximum de dojo pour le remplir de tatamis 2 x 1. La taille maximale est ([Longueur], [Largeur])". **Interprétation**: La taille (plus petite que les dimensions entrées) maximale d'un dojo pour obtenir une solution est de longueur [Longueur] et de largeur [Largeur].

   (b) **Réponse**: "Demande impossible. Il existe au moins une disposition possible avec des tatamis 2x1 pour ce dojo". **Interprétation**: la demande est non pertinente car il existe au moins une disposition possible de tatamis 2x1 pour les dimensions du dojo entrées.

8.3.9 En cas d'absence de solutions, existe-t-il une solution avec des demi-tatamis?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)
2. Entrer dans l'interface la longueur et la largeur du dojo dans les champs prévus à cet effet. (Champs au bas de l'interface intitulés "Longueur (entre 1 et 25)" et "Largeur (entre 1 et 25)").

   Nb:

   - *Vous pouvez inverser la longueur et la largeur, cela n'a pas d'importance*
   - *Vous pouvez entrer un nombre de tatamis compris entre 1 et 25.*
3. Cliquer sur le bouton: "Quelle taille maximum de dojo pour avoir une solution"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.
4. Interpréter la réponse :

   (a) **Réponse**: "Oui". **Interprétation**: Le dojo (aux dimensions entrées) ne peut pas être rempli par des tatamis de taille 2 x 1 mais il peut l'être en combinant au moins un demi-tatamis (de taille 1 x 1).

   (b) **Réponse**: "Demande impossible. Il existe au moins une disposition possible avec des tatamis 2x1 pour ce dojo". **Interprétation**: la demande est non pertinente car il existe au moins une disposition possible sans demi-tatamis pour les dimensions du dojo entrées.

8.3.10 Comment obtenir les tailles des dojos admettant des solutions de remplissage à partir d'un nombre de tatamis 2 x 1?

1. Lancer l'interface (dans le Terminal avec la ligne de commande: `python3 interface.py`)
2. Entrer dans l'interface le nombre de tatamis dans le champ prévu à cet effet (Champ au bas de l'interface intitulé "Nombre de tatamis 2 x 1 disponibles (entre 1 et 300)").

   Nb: Vous pouvez entrer un nombre de tatamis compris entre 1 et 300.
3. Cliquer sur le bouton: "Obtenir une solution étant donne un nombre de tatamis"

   Nb: Si vous oubliez d'entrer une dimension ou si vous entrez une dimension 0, un message d'erreur "Erreur de saisie des dimensions. Aucune dimension ne peut avoir une valeur nulle ou vide" apparaît.
4. Interpréter la réponse :

   (a) **Réponse**: "Dimension(s) possible(s) du dojo étant donné le nombre de tatamis saisis" suivi d'une liste de dimensions (couple de nombre de forme (longeur, largeur)). **Interprétation**: Les dimensions de dojos affichées sont des dimensions possibles de dojo permettant de remplir le dojo avec le nombre de tatamis 2 x 1 saisi.

   Nb: toutes les solutions de dimensions respectent:

- Un ratio maximum de 3 entre la longueur et la largeur
- Une utilisation au minimum de 75% du nombre de tatamis

(b) **Réponse vide** (“Dimension(s) possible(s) du dojo étant donné le nombre de tatamis saisi” sans dimensions affichées en dessous). **Interprétation**: il n’existe pas de dimensions de dojo permettant de le remplir avec le nombre de tatamis 2 x 1 saisi (et en respectant les conditions précédemment mentionnées).

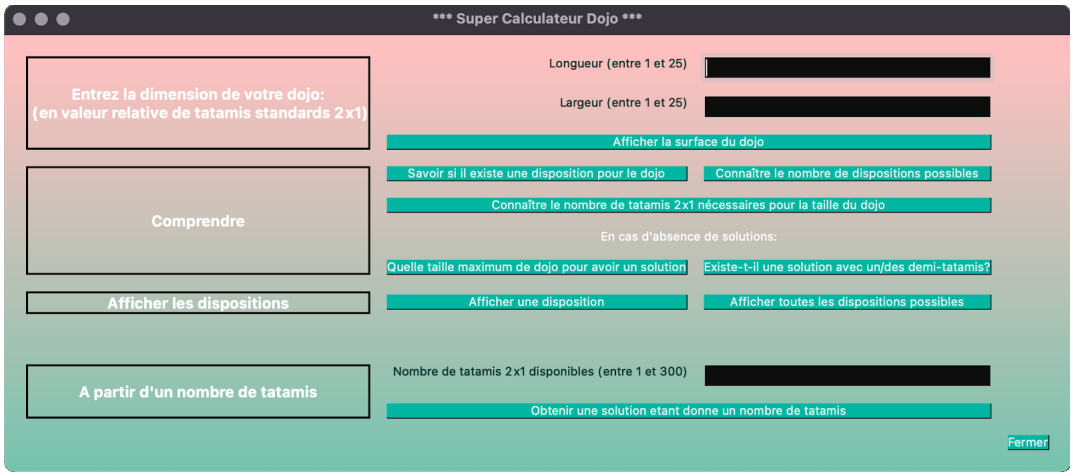
8.4 Explication de(s) algorithme(s) et choix de programmation

8.4.1 Algorithmes

8.4.2 Choix de programmation Interface

Outre l’ajout à l’interface des quatre boutons correspondants aux nouvelle fonctionnalités, les principaux changements ont été effectués à l’interface:

- Repositionnement des boutons et fonctionnalités: avec l’ajout de quatre fonctionnalités pour cette version, la compréhension commence à devenir compliquée pour l’utilisateur. Pour faciliter la navigation et l’utilisation, les choix suivants ont été fait:
  - Création de blocs visuel pour regrouper les fonctionnalités qui se rapprochent (en utilisant l’encadrement des textes pour arriver à l’effet souhaité)
  - Séparation visuelle de la fonctionnalité ajouté nécessitant la saisie d’une nouvelle donnée (en ajoutant une ligne vide dans la grille)
- Travail graphique pour améliorer le rendu visuel et utilisation des possibilités offertes par PyQt5
  - Couleur du fond avec un dégradé de couleurs
  - Couleur et forme des boutons
  - Changement de la police et modification des couleurs des textes



8.4.3 Choix de la structure du programme

Dans la continuité de l'application des principes *SOLID*, en particulier la volonté qu'un fichier ait une fonction principale, un fichier supplémentaire a été créé `release.py`. L'objectif de ce fichier est d'héberger les fonctionnalités implémentées spécifiquement pour cette version, à savoir :

- le calcul de solutions pour des dimensions plus petites en cas d'impossibilité de pavage avec les premières dimensions fournies par l'utilisateur.
- la fonction de calcul permettant de connaître les dimensions possibles de dojos pour un nombre de tatamis donné.

8.5 Challenges rencontrés et apprentissage

8.5.1 Challenges rencontrés et solutions appliquées

Les deux challenges principaux de cette version ont été les suivants:

- Charge de travail du sprint. Malgré avoir réalisé lors du précédent Sprint l'importance de ne pas sous-estimer la charge de travail pour un sprint, nous avons eu une nouvelle fois des difficultés dans cet exercice. Nous avons pourtant essayé de bien planifiée (cf. partie suivante 'Apprentissage') mais avec 4 fonctionnalités et des travaux sur l'interface, nous avons encore une fois été très ambitieux. La raison est en particulier due au fait que 2 des fonctionnalités ont nécessité des efforts importants sur l'algorithme.
- Challenges techniques pour les fonctionnalités:
  - Taille maximum d'un dojo ayant des disposition(s) possible(s) (en cas d'absence de solution pour le dojo aux dimensions saisies par l'utilisateur)
  - Obtention de dimensions de dojos (avec solutions) étant donné un nombre de tatamis saisis par l'utilisateur. Cette fonctionnalité a été particulièrement difficile car les possibilités sont multiples et des critères de sélection ont dû être ajoutés. Une recherche de doublons a également dû être appliquée.

8.5.2 Apprentissage

Le principal apprentissage est la phase de préparation et priorisation des User Stories à *embarquer* dans la version.

Étant donné les challenges rencontrés lors du précédent Sprint, nous avons décidé de mieux planifier les 2 derniers Sprint et avons procédé ainsi:

- Liste des fonctionnalités/User stories potentielles
- Travaux de recherche et estimation de la complexité selon 2 dimensions: effort sur l'algorithme et effort sur l'interface. Efforts dimensionnés en 3 catégories: faible, moyen et élevé
- Décisions sur les User stories pour les 2 derniers Sprint en essayant d'équilibrer la charge de travail.

Bien que nous ayons à nouveau sur-dimensionné le Sprint Release Candidate, nous sommes satisfait de la méthodologie de planification appliquée. Notre erreur se trouve au niveau de l'estimation de l'effort, probablement dû à notre manque d'expérience de la programmation. Pour le dernier Sprint, nous essayerons de compenser notre manque d'expérience par une réflexion plus approfondie du travail à effectuer pour achever la User Story.

9. Version Production

9.1 Backlog/roadmap du sprint Production

L'objectif de ce sprint est de finaliser en augmentant encore la valeur donné à l'utilisateur par: Ajout de deux fonctionnalités concernant les dispositions symétriques. L'objectif étant d'éliminer ces dispositions symétriques qui apportent peu, voire prêtent à confusion pour l'utilisateur. Dernière amélioration mineure de l'interface, surtout sur le visuel, pour aligner le design des messages de réponse à celui de l'interface principale.

Pour ce faire le backlog suivant a été *embarqué* dans cette version et a résulté en la roadmap suivante:

9.2 Tests

Les tests réalisés pour cette version et leurs résultats sont les suivants:

id	Sujet	Test d'acceptance	Méthode de test	Résultat
523	US Comprendre le nombre de dispositions possibles excluant les dispositions symétriques	Étant donné que l'utilisateur saisie des dimensions valides de dojo, il obtient le nombre de solutions possibles en excluant les solutions symétriques	Manuel	OK
524	US Afficher visuellement toutes les dispositions possibles excluant les dispositions symétriques	Étant donné des dimensions d'un dojo saisies, quand il sélectionne cette option, il obtient visuellement toutes les dispositions possibles en excluant les solutions symétriques	Manuel	OK

Par ailleurs, étant donné qu'il s'agit de la dernière version pour mise en production, il est important de tester l'ensemble des fonctionnalités du programme. Des tests de non régression ont été effectués:

Erreur sur validation des saisies:
Une erreur a été constaté suivant les versions utilisés de PyQt:

- Version 5.9.7: la validation de la saisie des données fonctionne (comportement attendu)

- Version 5.12.2: la validation fonctionne partiellement (cela limite bien le type d'entrée - uniquement des entiers, et limite le nombre de chiffres, mais cela ne limite pas à 25 ou 300 comme attendu; cela limite uniquement à respectivement 99 et 999). Pour corriger ce changement de comportement de QIntValidator, un nouveau message d'erreur a été mis en place.

9.3 Documentation Utilisateur Production

9.3.1 Prérequis

Configuration et installations requises:

- Python 3.9 ou supérieur
- Librairies Python: datetime, numpy, PyQt5.QtCore, PyQt5.QtGui, PyQt5.QtWidgets, sys, time. (Idéalement la version 5.9.7 de PyQt)
- Installer les polices Nexa (light et bold) (fichiers otf fournis)