Table des matières

[Étape 1 : Définir les caractéristiques du Tamagotchi 4](#_Toc155475169)

[1. Quelles sont les caractéristiques de base du Tamagotchi ? 4](#_Toc155475170)

[2. Quelles sont les actions que l'utilisateur peut effectuer sur le Tamagotchi ? 4](#_Toc155475171)

[3. Comment évolue le Tamagotchi au fil du temps ? 4](#_Toc155475172)

[4. Quels sont les états possibles du Tamagotchi ? 4](#_Toc155475173)

[5. Quels sont les facteurs qui influencent ces caractéristiques ? 5](#_Toc155475174)

[6. Quels sont les objectifs du jeu ? 5](#_Toc155475175)

[7. Avez-vous des idées pour des fonctionnalités spéciales ou des surprises pour rendre votre Tamagotchi unique ? 5](#_Toc155475176)

[8. Quelle sera l'interface utilisateur pour permettre à l'utilisateur d'interagir avec le Tamagotchi ? 5](#_Toc155475177)

[Documentation du Code Etape 1: Tamagotchi en C++ 6](#_Toc155475178)

[Aperçu Général 6](#_Toc155475179)

[Structure du Code 6](#_Toc155475180)

[Classe Tamagotchi 6](#_Toc155475181)

[Méthodes : 6](#_Toc155475182)

[Fonction « jouerTamagotchi » 7](#_Toc155475183)

[Fonction « main » 7](#_Toc155475184)

[Choix de Conception et Alternatives 7](#_Toc155475185)

[Conclusion 7](#_Toc155475186)

[Étape 2 : Création de la classe Tamagotchi 8](#_Toc155475187)

[1.Inclure les fichiers d'en-tête nécessaires 8](#_Toc155475188)

[2. Définir la classe Tamagotchi 8](#_Toc155475189)

[3. Définir le constructeur 9](#_Toc155475190)

[4. Implémenter les méthodes 9](#_Toc155475191)

[5. Définir des méthodes d'accès 10](#_Toc155475192)

[6. Gérer la sécurité de la classe 10](#_Toc155475193)

[Conclusion 11](#_Toc155475194)

[Étape 3 : Définir les méthodes pour la classe Tamagotchi 12](#_Toc155475195)

[1. Méthode manger() 12](#_Toc155475196)

[2. Méthode dormir() 12](#_Toc155475197)

[3. Méthode laver() 12](#_Toc155475198)

[4. Méthode soigner() 13](#_Toc155475199)

[4 bis. Gérez les limites 13](#_Toc155475200)

[Conclusion 14](#_Toc155475201)

[Étape 4 : Mettre en place une boucle de jeu 15](#_Toc155475202)

[1. Initialisation du Tamagotchi 15](#_Toc155475203)

[2. Boucle principale 16](#_Toc155475204)

[3. Affichage de l'état et interaction utilisateur 16](#_Toc155475205)

[4. Mise à jour de l'état du Tamagotchi 17](#_Toc155475206)

[5. Vérification de la fin de jeu 18](#_Toc155475207)

[6. Gestion de la fin de jeu 19](#_Toc155475208)

[Conclusion 19](#_Toc155475209)

[Étape 5 : Interface utilisateur textuelle 20](#_Toc155475210)

[1. Afficher l'état du Tamagotchi 20](#_Toc155475211)

[2. Afficher un menu d'options 22](#_Toc155475212)

[3. Lire l'entrée de l'utilisateur 23](#_Toc155475213)

[4. Traitement des choix de l'utilisateur 24](#_Toc155475214)

[5. Validation des entrées utilisateur 25](#_Toc155475215)

[6. Boucle principale 26](#_Toc155475216)

[Conclusion 26](#_Toc155475217)

[Étape 6 : Gestion du temps par tours 27](#_Toc155475218)

[1.Définir le concept de tour 27](#_Toc155475219)

[2. Compteur de tours 27](#_Toc155475220)

[3. Ajouter une boucle de tour 28](#_Toc155475221)

[4. Passage de temps quotidien 28](#_Toc155475222)

[5. Gestion du vieillissement 29](#_Toc155475223)

[6. Gestion de la faim et du bonheur 29](#_Toc155475224)

[7. Événements aléatoires 30](#_Toc155475225)

[Conclusion 31](#_Toc155475226)

[Étape 7 : Gestion de l'état du Tamagotchi 32](#_Toc155475227)

[Définir les états possibles : 32](#_Toc155475228)

[Mise à jour de l'état : 32](#_Toc155475229)

[Afficher l'état du Tamagotchi : 33](#_Toc155475230)

[Réagir aux états critiques : 33](#_Toc155475231)

[Événements aléatoires basés sur l'état : 34](#_Toc155475232)

[Étape 8 : Gestion de la sauvegarde 35](#_Toc155475233)

[Créer une classe de gestion de la sauvegarde : 35](#_Toc155475234)

[Implémenter la sauvegarde : 35](#_Toc155475235)

[Implémenter le chargement : 35](#_Toc155475236)

Projet : Création d'un Tamagotchi en C++

# Étape 1 : Définir les caractéristiques du Tamagotchi

Pour commencer votre projet, il est essentiel de bien comprendre les caractéristiques et les fonctionnalités que votre Tamagotchi virtuel devrait posséder. Cette étape de conception est cruciale pour définir clairement les besoins et les attentes de votre projet. Voici quelques questions à prendre en compte pour vous aider à définir ces caractéristiques :

## 1. Quelles sont les caractéristiques de base du Tamagotchi ?

* Santé : Un indicateur numérique qui diminue avec le temps ou lors d'événements négatifs.
* Faim : Un indicateur qui augmente avec le temps, nécessitant une alimentation régulière.
* Humeur : Varie selon les interactions et le temps passé sans attention.

Ces caractéristiques sont essentielles pour simuler un animal de compagnie vivant. La santé représente le bien-être global, la faim la nécessité de soins réguliers, et l'humeur l'interaction avec l'utilisateur.

## 2. Quelles sont les actions que l'utilisateur peut effectuer sur le Tamagotchi ?

* Nourrir : Réduit l'indicateur de faim.
* Jouer : Améliore l'humeur.
* Soigner : Améliore la santé si elle est dégradée.

Chaque action influe directement sur une caractéristique du Tamagotchi, créant une expérience interactive.

## 3. Comment évolue le Tamagotchi au fil du temps ?

Avec le temps, il peut grandir, changer d'aspect ou de comportement. Les actions de l'utilisateur peuvent accélérer ou ralentir cette évolution.Cela ajoute un élément de progression et récompense l'engagement de l'utilisateur.

## 4. Quels sont les états possibles du Tamagotchi ?

Heureux, Triste, Malade, etc. Ces états dépendent des caractéristiques de base.Varier les états rend le jeu plus réaliste et engageant.

## 5. Quels sont les facteurs qui influencent ces caractéristiques ?

* Comportement de l’Utilisateur : Les interactions de l'utilisateur avec le Tamagotchi, comme le nourrir, jouer avec lui ou le soigner, influencent directement ses caractéristiques comme la santé, le bonheur, et la faim.
* Temps Réel : Le temps qui s'écoule est un facteur clé. Par exemple, si le Tamagotchi n'est pas nourri ou soigné pendant un certain temps, ses niveaux de faim et de santé pourraient se dégrader.
* Événements Aléatoires : Des événements aléatoires peuvent être programmés pour affecter les caractéristiques du Tamagotchi, comme des maladies ou des découvertes de nourriture, ajoutant de l'incertitude et du défi au jeu.
* Étapes de Développement : Le Tamagotchi pourrait évoluer ou changer de phases au cours de son développement, influençant ses besoins et comportements.
* Interactions avec l'Environnement : Des éléments externes comme la météo ou des événements saisonniers pourraient influencer les caractéristiques du Tamagotchi.

## 6. Quels sont les objectifs du jeu ?

Déterminer les objectifs du jeu est crucial pour guider le développement. Ceci inclut les buts à court et long terme pour l'utilisateur, comme maintenir le Tamagotchi en vie, le faire évoluer, atteindre certains niveaux, etc. Ces objectifs influencent directement la conception des caractéristiques et des mécanismes du jeu.

## 7. Avez-vous des idées pour des fonctionnalités spéciales ou des surprises pour rendre votre Tamagotchi unique ?

Intégrer des fonctionnalités uniques ou des surprises peut augmenter l'engagement de l'utilisateur. Cela pourrait être des événements spéciaux, des interactions cachées, ou des récompenses uniques. Ces éléments doivent être bien pensés pour enrichir l'expérience sans compliquer outre mesure le code.

## 8. Quelle sera l'interface utilisateur pour permettre à l'utilisateur d'interagir avec le Tamagotchi ?

La conception de l'interface utilisateur est primordiale. Elle doit être intuitive et réactive, permettant à l'utilisateur d'interagir facilement avec le Tamagotchi. En C++, cela peut impliquer l'utilisation de bibliothèques graphiques ou de la console pour les entrées/sorties.

Une fois ces aspects définis, vous pouvez passer à la création de la classe Tamagotchi et à l'implémentation des fonctionnalités en utilisant la programmation orientée objet en C++. Chaque caractéristique ou fonctionnalité devrait être encapsulée dans des méthodes et des attributs

# Documentation du Code Etape 1: Tamagotchi en C++

## Aperçu Général

Ce code représente une simulation simple d'un Tamagotchi, un animal de compagnie virtuel. Le programme permet à l'utilisateur de nourrir, jouer avec, soigner le Tamagotchi, et de vérifier son statut. Chaque action influe sur les indicateurs de faim, de bonheur, et de santé du Tamagotchi. Si l'un de ces indicateurs tombe à zéro, le Tamagotchi meurt.

## Structure du Code

Inclusions et Utilisation de Namespace

#include <iostream> : pour les entrées/sorties standard.

#include <string>: pour utiliser des objets de type « std::string ».

## Classe Tamagotchi

« class Tamagotchi » : Définit les caractéristiques et comportements d'un Tamagotchi.

Attributs privés: « nom », « faim », « sante », « bonheur », « estVivant » - Stockent les informations essentielles du Tamagotchi.

Constructeur : Initialise un Tamagotchi avec un nom et des valeurs par défaut pour la faim, la santé, et le bonheur.

## Méthodes :

« nourrir », « jouer », « soigner » : Augmentent les indicateurs de faim, de bonheur, et de santé respectivement.

« statut » : Affiche l'état actuel du Tamagotchi.

« estEnVie »: Retourne l'état de vie du Tamagotchi.

« miseAJour » : Diminue les indicateurs de faim, bonheur et santé à chaque appel, et vérifie si le Tamagotchi est toujours en vie.

## Fonction « jouerTamagotchi »

- Gère le jeu interactif avec le Tamagotchi, en permettant à l'utilisateur de choisir différentes actions.

- Utilise « std::cin » pour les entrées de l'utilisateur et gère le flux d'entrée pour éviter les erreurs de saisie.

- Emploie une boucle « while » pour continuer le jeu tant que le Tamagotchi est en vie.

- Utilise `system("cls")` ou `system("clear")` pour nettoyer l'écran. Une alternative serait d'utiliser des bibliothèques multiplateformes pour le nettoyage de l'écran.

## Fonction « main »

- Point d'entrée du programme.

- Permet à l'utilisateur de jouer plusieurs fois avec des Tamagotchis différents en utilisant une boucle « do-while ».

## Choix de Conception et Alternatives

Gestion des états du Tamagotchi : Les indicateurs de faim, bonheur, et santé sont gérés simplement par des entiers. Une approche plus avancée pourrait utiliser des structures ou des classes distinctes pour gérer ces états de manière plus détaillée.

Gestion des entrées utilisateur : L'utilisation de « std::cin » et « std::cin.ignore() » est adéquate mais peut être remplacée par des méthodes plus robustes pour gérer les entrées incorrectes.

Portabilité : L'utilisation de « system("cls") » ou « system("clear") » est dépendante du système d'exploitation, ce qui réduit la portabilité du code. Il est préférable d'utiliser des méthodes multiplateformes pour une meilleure compatibilité.

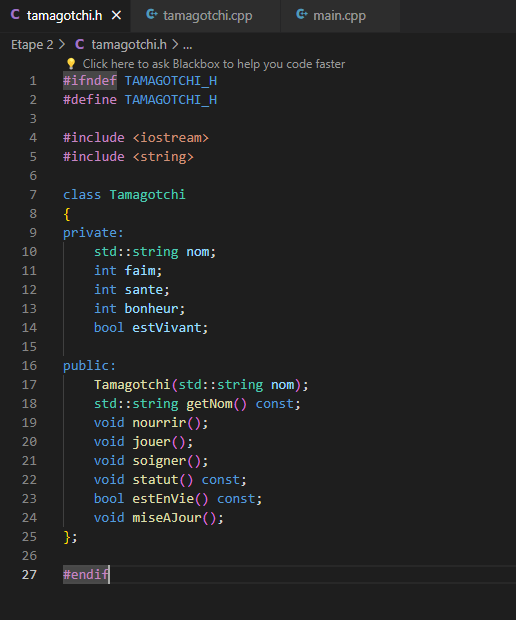
## Conclusion

Le code fournit une base solide pour une simulation de Tamagotchi en C++, avec une structure claire et des fonctionnalités de base. Cependant, il existe des possibilités d'amélioration en termes de gestion des états, de la robustesse des entrées et de la portabilité.

# Étape 2 : Création de la classe Tamagotchi

## 1.Inclure les fichiers d'en-tête nécessaires

Commencez par inclure dans votre header (.h) les fichiers d'en-tête nécessaires, tels que pour la gestion de la saisie/sortie et si vous prévoyez d'utiliser des chaînes de caractères. Il faudra également inclure votre fichier ".h" (tamagotchi.h), en en-tête de votre ".cpp".



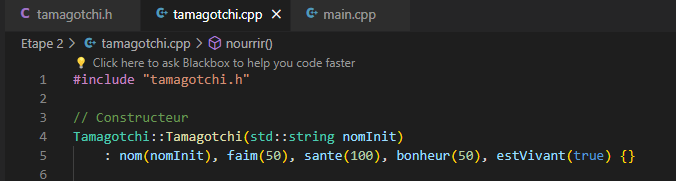
## 2. Définir la classe Tamagotchi

Dans le header de la classe (tamagotchi.h), créez une classe Tamagotchi en utilisant le mot-clé class. Déclarez les membres de la classe qui représenteront les caractéristiques du Tamagotchi.

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

3. Définir le constructeur

Créez un constructeur pour la classe Tamagotchi. Le constructeur devrait initialiser les caractéristiques de base du Tamagotchi lors de sa création.

## 4. Implémenter les méthodes

Écrivez les méthodes définies dans la classe, telles que manger(), jouer(), dormir(), etc. Ces méthodes devraient modifier les caractéristiques du Tamagotchi en conséquence.

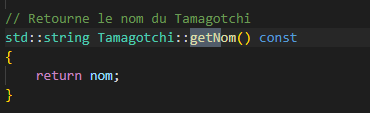
Une image contenant texte, capture d’écran

Description générée automatiquement

## 5. Définir des méthodes d'accès

Pour permettre à d'autres parties de votre code d'accéder aux caractéristiques du Tamagotchi, créez des méthodes d'accès (getters) appropriées.

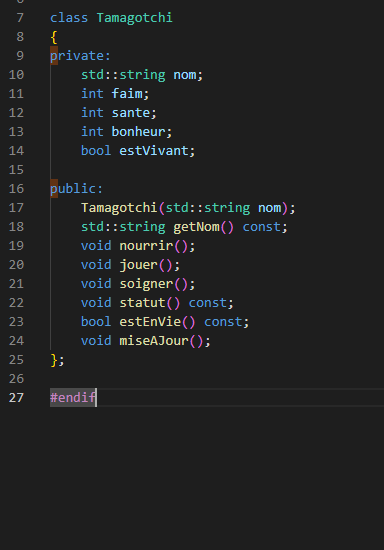
Dans le fichier « tamagotchi.cpp », la méthode d'accès (getter) pour obtenir le nom du Tamagotchi est définie comme suit :



Cette méthode « getNom » permet d'accéder à la valeur privée « nom » de l'objet « Tamagotchi ». Elle est marquée « const », indiquant qu'elle ne modifie pas l'état de l'objet. Cette méthode retourne une copie de la valeur de l'attribut « nom ».

## 6. Gérer la sécurité de la classe

Vous pouvez utiliser des mécanismes d'encapsulation pour protéger les membres de la classe et éviter un accès direct non autorisé. (public/protected/private)



Explication de l'Encapsulation :

* Membres Privés (private)

Les attributs « nom », « faim », « sante », « bonheur », et « estVivant » sont déclarés comme privés. Cela signifie qu'ils ne peuvent être accédés ou modifiés directement en dehors de la classe « Tamagotchi ». Cette approche protège l'état interne de l'objet « Tamagotchi » contre les modifications non autorisées ou accidentelles, et elle permet de maintenir l'intégrité de l'objet.

* Méthodes et Constructeur Publics (public )

Le constructeur « Tamagotchi(std::string nom) » et les méthodes « getNom », « nourrir », « jouer », « soigner », « statut », « estEnVie », et « miseAJour » sont déclarés comme publics. Ces méthodes fournissent un accès contrôlé aux attributs privés de la classe. Par exemple, « getNom » permet de récupérer le nom du Tamagotchi, mais ne permet pas de le modifier.

Cette structure assure que l'interaction avec les attributs de la classe `Tamagotchi` ne peut se faire que de manière contrôlée et intentionnelle, à travers les méthodes publiques.

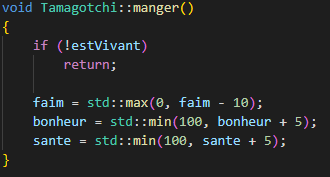
## Conclusion

Dans cette deuxième étape du projet Tamagotchi, j'ai réalisé d'importantes améliorations. En mettant en œuvre l'encapsulation, j'ai assuré la sécurité des données de la classe « Tamagotchi », rendant l'accès et la modification des attributs de l'objet contrôlés et sécurisés. J'ai également clarifié la structure du code en séparant les définitions et les implémentations, améliorant ainsi la lisibilité et la maintenabilité. De plus, l'introduction de fonctionnalités interactives a enrichi l'expérience de jeu, rendant le Tamagotchi plus dynamique et engageant. Ces avancées ont non seulement élevé la qualité du jeu, mais ont également renforcé mes compétences en programmation orientée objet en C++.

# Étape 3 : Définir les méthodes pour la classe Tamagotchi

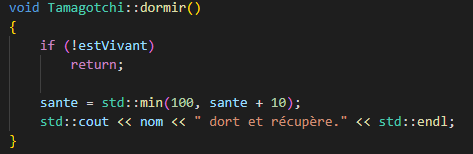
# 

1. Méthode manger()  
Cette méthode permettra au Tamagotchi de manger et de satisfaire sa faim. Elle devrait modifier la jauge de faim du Tamagotchi et potentiellement modifier d'autres caractéristiques comme le bonheur, la santé, l’hygiène, … en fonction de la logique du jeu.



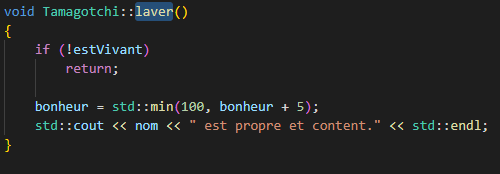
## 2. Méthode dormir()

Cette méthode permettra au Tamagotchi de se reposer et d'augmenter son énergie (ou diminuer sa fatigue selon votre projet). Elle peut également affecter d'autres caractéristiques.



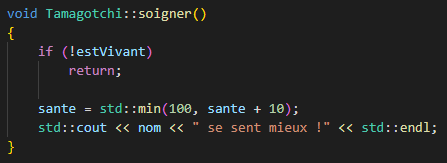
3. Méthode laver()

Cette méthode permettra au Tamagotchi d’être lavé et d'augmenter son hygiène. Elle peut également affecter d'autres caractéristiques.



## 4. Méthode soigner()

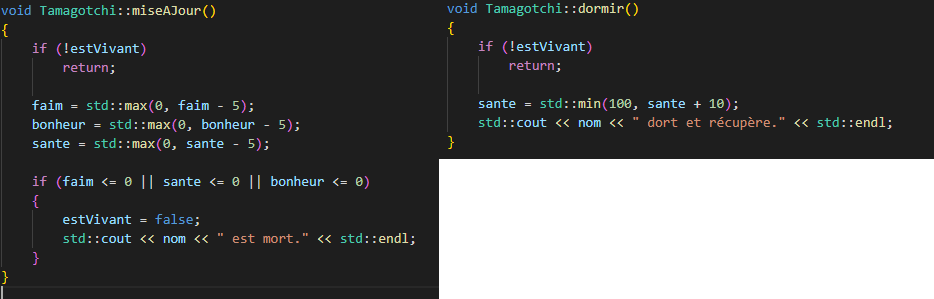
Cette méthode permettra au Tamagotchi d’être soigné, et ainsi d’augmenter sa santé. Elle peut également affecter d'autres caractéristiques.



## 4 bis. Gérez les limites

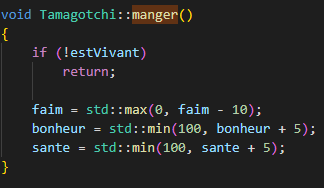
Assurez-vous de gérer les limites des caractéristiques pour éviter qu'elles ne dépassent les valeurs valides (par exemple, 0 à 100). Vous pouvez utiliser des instructions if pour vérifier et ajuster les valeurs si nécessaire.

Pour s'assurer que les valeurs restent entre 0 et 100 dans mon code, j’ai utilisé les fonctions « std::max » et « std::min » de la bibliothèque standard C++. Voici les parties du code où ces fonctions sont utilisées :



Ces lignes s'assurent que les valeurs de `faim`, `bonheur`, et `sante` ne descendent jamais en dessous de 0, même après avoir soustrait une certaine quantité. Dans le cas échéant, le tamagotchi meurt.

Pour maintenir la valeur à un maximum de 100 j’ai utilisé les mêmes fonctions que précédemment



Cet exemple illustre le cas de la méthode manger(), mais c’est aussi présent dans soigner(), jouer(), dormir() et laver().

Ces lignes s'assurent que les valeurs de « bonheur » et « sante » ne dépassent jamais 100, même après avoir ajouté une certaine quantité. Dans le cas échéant, si le score atteint 100, rien ne se passe, car dans les Tamagotchi, il n’y pas de condition de réussite.

## Conclusion

Dans cette nouvelle étape, j'ai implémenté des contrôles de limite pour assurer que les attributs comme la faim, le bonheur et la santé restent entre 0 et 100. J'ai utilisé les fonctions « std::max » et « std::min » pour cela. La principale difficulté rencontrée était de s'assurer que ces contrôles fonctionnent correctement dans toutes les situations, en particulier lors de la mise à jour de l'état du Tamagotchi. Cette étape était cruciale pour éviter des valeurs négatives ou supérieures aux limites fixées, qui auraient pu causer des comportements inattendus dans le jeu.

# Étape 4 : Mettre en place une boucle de jeu

Dans cette étape, je n’ai que peu ajouté de modification car beaucoup car beaucoup de règle du jeu était déjà mise en place pour que je puisse effectuer les tests correctement.

## 1. Initialisation du Tamagotchi

Tout d'abord, créez une instance de la classe Tamagotchi pour représenter votre Tamagotchi virtuel. Vous pouvez demander à l'utilisateur de donner un nom à son Tamagotchi ou le générer aléatoirement.

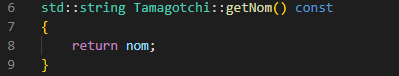
Les lignes de code qui correspondent cette exigence ,l'initialisation du Tamagotchi, se trouvent dans deux parties de mon programme :

* Le Constructeur de la Classe Tamagotchi: Ce constructeur initialise un nouvel objet Tamagotchi avec des valeurs par défaut pour la faim, la santé, le bonheur, et le statut de vie. Voici les lignes pertinentes :



Ce constructeur est appelé lorsque je crée une nouvelle instance de Tamagotchi, initialisant les valeurs de base de l'objet.

* Création d'une Instance dans la Fonction « jouerTamagotchi »: Dans ma fonction principale qui gère le jeu, je crée une nouvelle instance de Tamagotchi en demandant à l'utilisateur de saisir un nom. Voici les lignes pertinentes :



Ici, l'utilisateur est invité à entrer un nom pour son Tamagotchi, et une nouvelle instance de

Tamagotchi est créée avec ce nom.

## 2. Boucle principale

Créez une boucle qui continuera tant que le Tamagotchi est en vie et que l'utilisateur souhaite continuer à jouer. Vous pouvez utiliser une boucle while pour cela.

Ici, la seule condition de la boucle infinie est que le Tamagotchi sois toujours en vie. La déclaration de la seule condition est donc très courte.



## 3. Affichage de l'état et interaction utilisateur

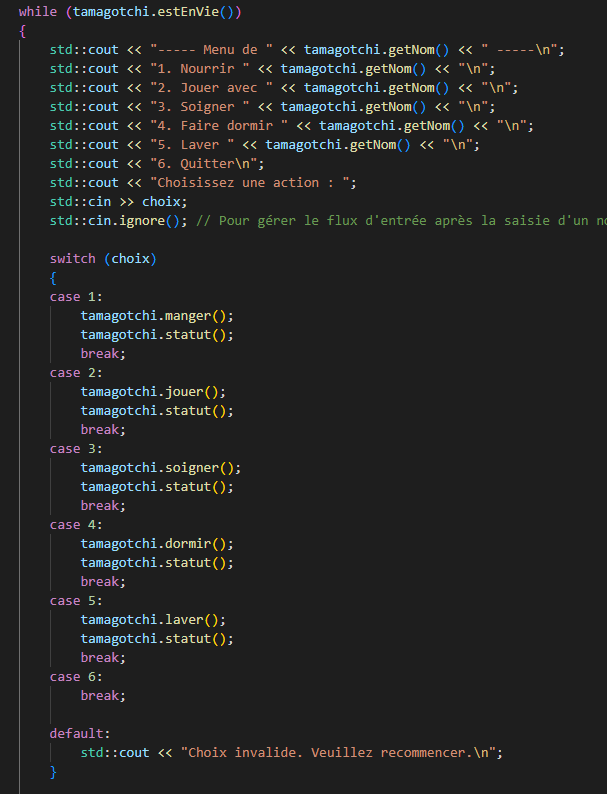
À l'intérieur de la boucle, vous devrez afficher l'état actuel du Tamagotchi (niveau de faim, bonheur, etc.) et demander à l'utilisateur ce qu'il souhaite faire en affichant un menu. L'utilisateur devrait pouvoir choisir parmi différentes actions, telles que nourrir, jouer, dormir, etc.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

## 4. Mise à jour de l'état du Tamagotchi

En fonction du choix de l'utilisateur, appelez les méthodes appropriées du Tamagotchi pour mettre à jour ses caractéristiques. Par exemple, si l'utilisateur choisit de le faire manger, appelez tamagotchi.manger().



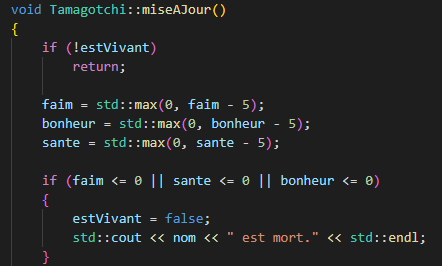
Ici les mises à jour des statu respectifs sont effectuées dans chaque méthode. Le menu est affiché et l’utilisateur fait son choix. A l’aide de switch() et case, le choix correspond a l’exécution d’une des méthodes.

## 5. Vérification de la fin de jeu

À chaque tour de boucle, vérifiez si le Tamagotchi est toujours en vie. Si ses caractéristiques atteignent un état critique (par exemple, faim ou bonheur très bas, santé très basse), vous pouvez considérer le Tamagotchi comme mort et mettre fin au jeu.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement



C’est la combinaison d’un booléen et de la méthode miseAJour qui permet de toujours vérifier l’état de vie du Tamagotchi.

## 6. Gestion de la fin de jeu

Lorsque le jeu se termine (par exemple, le Tamagotchi meurt ou l'utilisateur choisit de quitter), affichez un message approprié et terminez la boucle de jeu.

Dans le cas où l’utilisateur quitte le jeu avec le choix 6, un booléen change d’état. Ce booléen est une des conditions de la boucle infinie.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Dans le cas où l’un des statuts tombe à 0, un message annonçant la mort du Tamagotchi est affiché. Je propose donc au joueur de recommencer à jouer, ou de quitter.



## Conclusion

En conclusion, peu de modifications ont été nécessaires pour répondre aux besoins de cette phase du projet. Cependant, j'ai intégré un élément important : l'ajout de l'option de sortie, le choix 6, dans le menu du jeu. Ce changement permet au joueur de quitter le jeu de manière intentionnelle et contrôlée. Bien que cette modification soit relativement simple, elle améliore l'interactivité et la convivialité du jeu, en donnant aux utilisateurs la flexibilité de terminer leur session de jeu quand ils le souhaitent.

# Étape 5 : Interface utilisateur textuelle

## 1. Afficher l'état du Tamagotchi

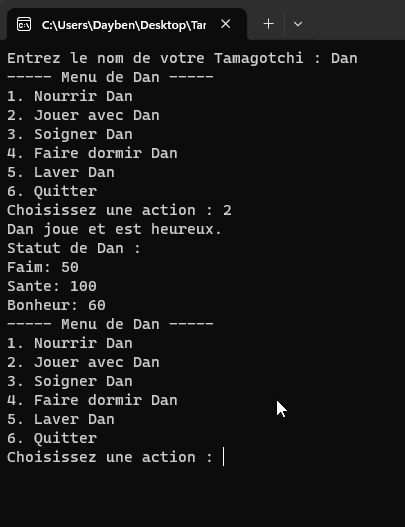
À chaque tour de boucle, affichez l'état actuel du Tamagotchi pour que l'utilisateur puisse voir ses caractéristiques telles que la faim, le bonheur, la santé…

J’ai déjà implémenté cette fonction. A chaque action, le statut du Tamagotchi est affiché. Cela permet de garder un œil constamment sur son état, et savoir si nous approchons bientôt la mort de ce dernier.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Cela donne cet affichage dans la console :



On retrouve bien l’affichage de son statut.

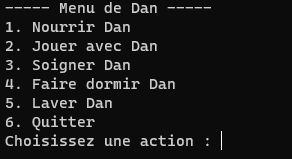
## 2. Afficher un menu d'options

Proposez à l'utilisateur un menu d'options qu'il peut choisir en entrant un numéro. Vous devriez utiliser un switch dans une fonction pour afficher ce menu. Ajoutez autant de lignes que nécessaire pour afficher toutes les actions possibles.

L’affichage est déjà présent, c’est une symbiose de « cout » et de « switch case » à fin d’afficher un menu, que nous verrons plus tard.

Une image contenant texte, capture d’écran, logiciel

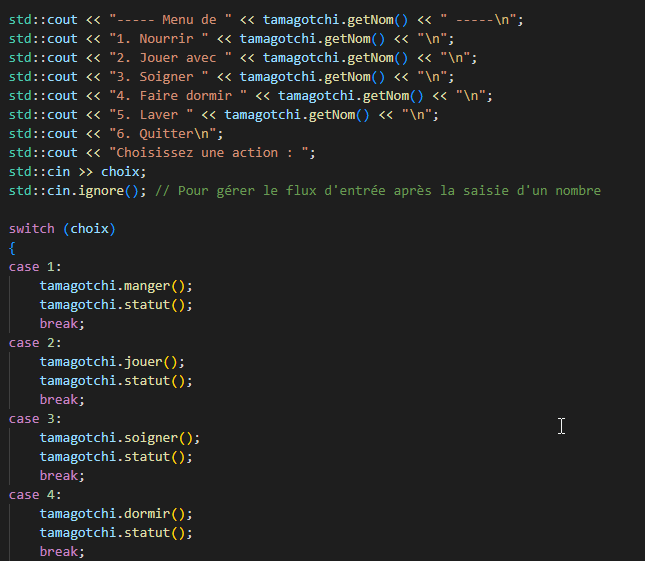
Description générée automatiquement



## 3. Lire l'entrée de l'utilisateur

Utilisez cin pour lire l'entrée de l'utilisateur, c'est-à-dire le numéro correspondant à son choix. Il faut créer une variable pour stocker ce choix.

C’est la suite de l’exercice précédent. Une variable « choix » récupère à l’aide d’un « cin » le choix de l’utilisateur. Ensuite le chiffre dans la variable « choix » est comparée à l’aide de la fonction switch(), et qui dans le cas ou la variable est valide, exécute les méthodes qui sont lié au choix.



## 4. Traitement des choix de l'utilisateur

Utilisez une structure de contrôle, telle qu'une instruction switch, pour exécuter le code approprié en fonction du choix de l'utilisateur. Dans cet exemple c’est la variable choixUtilisateur qui contient cette information. On la passe donc en paramètre du switch :

Une image contenant texte, capture d’écran

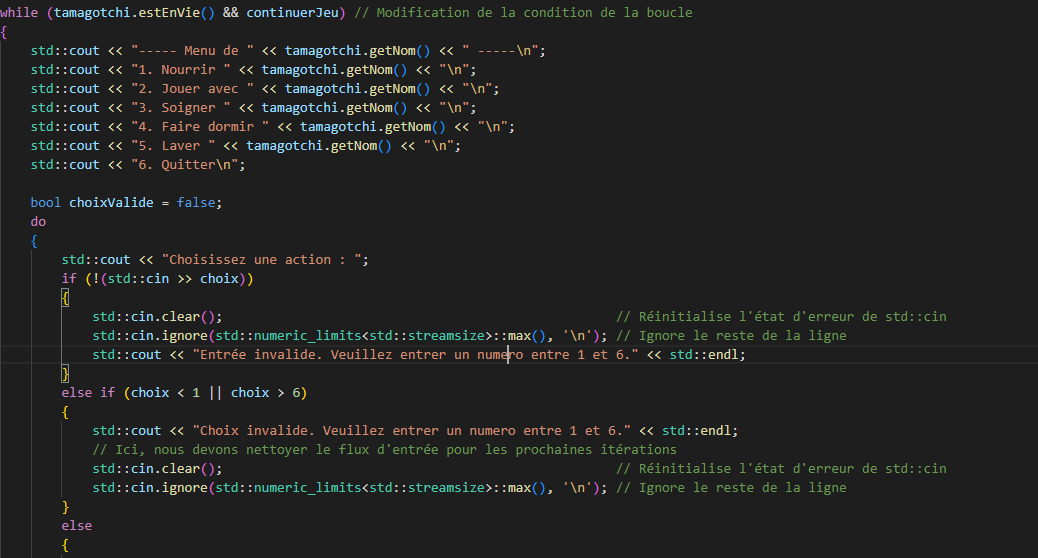
Description générée automatiquement

Le chiffre entré par l’utilisateur et précédemment stockerdans la variable « choix » est comparée à l’aide de la fonction switch(), et qui dans le cas où la variable est valide, exécute les méthodes qui sont lié au choix.

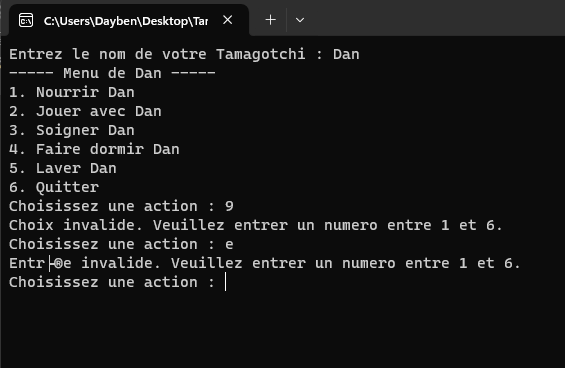
## 5. Validation des entrées utilisateur

Ajoutez des mécanismes de validation pour vous assurer que les entrées de l'utilisateur sont valides. Vous pouvez utiliser des boucles pour demander à l'utilisateur de réessayer en cas d'entrée incorrecte.

Nous avons un nouveau booléen. Celui-ci vérifié si le choix fait par l’utilisateur est valide. Si c’est le cas, alors la valeur est passée en paramètre du bloc de switch(). Pour éviter certaines erreurs, on vide la variable stocker dans « choix ».



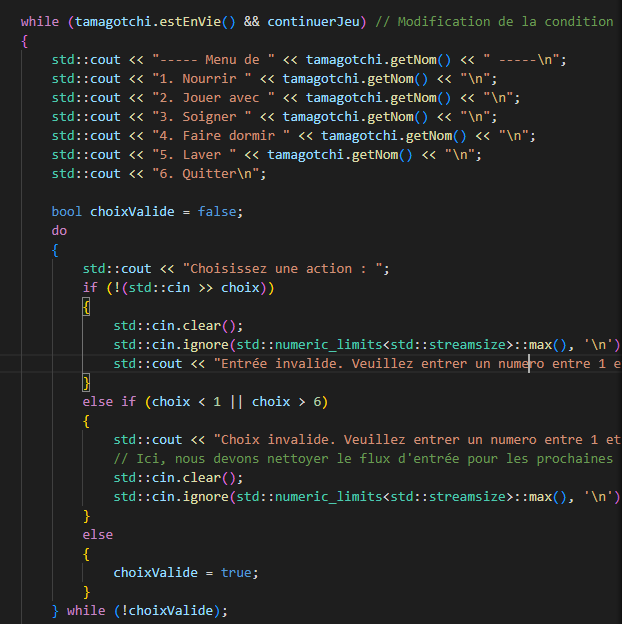
Voici le résultat, dans les deux cas le choix n’est pas valide, et l’utilisateur dois rentrer a nouveau un choix.



## 6. Boucle principale

Veillez à ce que cette interface utilisateur soit située à l'intérieur de votre boucle de jeu principale pour permettre à l'utilisateur d'interagir continuellement avec le Tamagotchi jusqu'à ce que le jeu se termine.

Nous avons deux boucles. La première vérifie que le tamagotchi est en vie, et que le l’utilisateur veuille bien encore jouer. La deuxième vérifie que le choix soit bien valide. Dans ces boucles nous avons bien le menu afficher a l’utilisateur, et ensuite les actions qui découle de son choix.



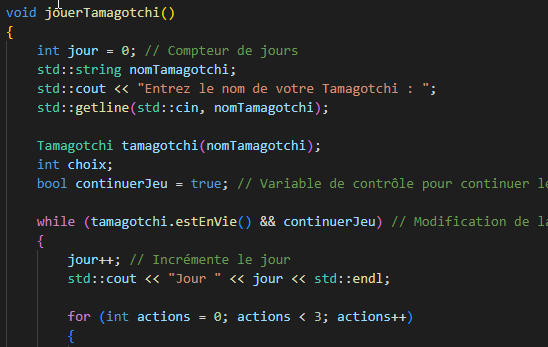
## Conclusion

En conclusion, j'ai apporté des modifications ciblées pour résoudre les problèmes de saisie utilisateur dans le jeu Tamagotchi. La principale difficulté rencontrée était liée à la demande répétée du choix de l'utilisateur, due à une structure de code redondante qui sollicitait l'entrée à deux reprises. En éliminant la première invitation de saisie avant la boucle « do-while », j'ai assuré que le choix ne soit demandé qu'une seule fois et validé correctement. J'ai également veillé à ce que les erreurs de saisie soient gérées avec soin, permettant ainsi à l'utilisateur de retenter sa chance sans interruption ni fermeture inattendue du jeu.

# Étape 6 : Gestion du temps par tours

## 1.Définir le concept de tour

Avant de commencer, déterminez ce que représente un tour dans votre jeu. Dans votre cas, un tour équivaut à un jour de vie pour le Tamagotchi, et vous pouvez définir un nombre d'actions possibles par tour (par exemple, 3 actions).



Ici, un "jour" dans la vie du Tamagotchi est représenté par une itération de la boucle while. La variable jour est utilisée pour compter le nombre de jours écoulés.

## 2. Compteur de tours

Créez une variable pour suivre le nombre de tours qui se sont écoulés depuis le début du jeu. Vous pouvez initialiser cette variable à zéro au début du jeu.   
  
Cette variable est présentée précédemment.

## 3. Ajouter une boucle de tour

À l'intérieur de votre boucle de jeu principale, ajoutez une autre boucle pour gérer les tours. Cette boucle s'exécutera tant que le jeu est en cours et permettra au Tamagotchi de vivre jour après jour.

Une image contenant texte, capture d’écran, Police

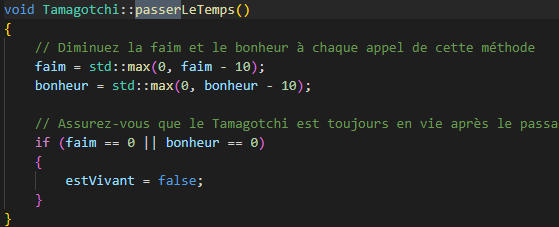
Description générée automatiquement

Cette boucle for exécute trois actions par jour, permettant à l'utilisateur de faire un nombre limité de choses avant la fin du jour.

## 4. Passage de temps quotidien

À la fin de chaque tour (un tour équivaut à un jour de vie), vous pouvez ajouter des mécanismes pour simuler le passage du temps quotidien. Par exemple, vous pouvez faire vieillir le Tamagotchi, augmenter sa faim, diminuer son bonheur, etc. Vous pouvez également ajouter des événements aléatoires qui se produisent au fil du temps.

Une image contenant texte, capture d’écran, Police

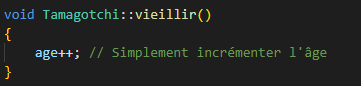
Description générée automatiquement

La méthode passerLeTemps() est appelée à la fin de chaque jour pour simuler l'effet du passage du temps sur les attributs du Tamagotchi.

## 5. Gestion du vieillissement

Si vous souhaitez que le Tamagotchi vieillisse, vous pouvez ajouter une variable d'âge à la classe Tamagotchi et l'incrémenter à chaque tour.

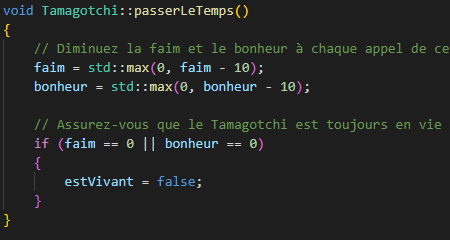
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La méthode vieillir() est appelée à la fin de chaque jour pour augmenter l'âge du Tamagotchi.

## 6. Gestion de la faim et du bonheur

Dans le passage de temps quotidien, vous pouvez augmenter la faim du Tamagotchi, réduire son bonheur, etc., pour simuler les besoins croissants du Tamagotchi.

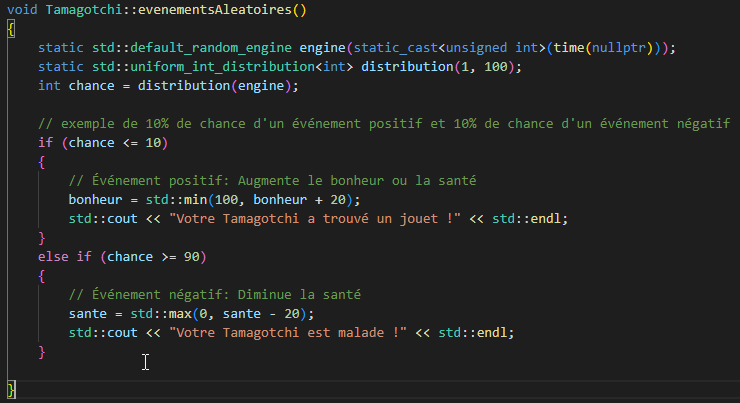


Ces lignes diminuent les niveaux de faim et de bonheur à la fin de chaque jour.

## 7. Événements aléatoires

Vous pouvez ajouter des événements aléatoires pour pimenter le jeu, par exemple, un jour de pluie qui affecte l'humeur du Tamagotchi ou une surprise positive qui le rend plus heureux. Ou encore une maladie aléatoire qui viendrait altérer la santé de votre animal virtuel.

Pour cette partie, je vais détailler ligne par ligne le code ci-dessous. Je vais introduire des notions non vues jusque-là.



« Static » : Indique que les objets « engine » et « distribution » sont initialisés une seule fois et conservent leur état entre les appels de la fonction. Cela permet de ne pas réinitialiser le générateur de nombres aléatoires à chaque appel, ce qui pourrait conduire à des résultats répétitifs si la fonction est appelée plusieurs fois rapidement.

« Std ::default\_random\_engine »: Il s'agit d'un générateur de nombres aléatoires fourni par la bibliothèque standard C++.

Static\_cast<unsigned int>(time(nullptr)) : Cette instruction sert à initialiser le générateur de nombres aléatoires avec un "seed" basé sur le temps actuel. Cela permet de s'assurer que les nombres générés sont différents à chaque exécution du programme.

« int chance »: Déclare une variable « chance » qui stockera le nombre aléatoire généré.

« distribution(engine) »: Génère un nombre aléatoire entre 1 et 100 inclus en utilisant le générateur « engine ». La variable « chance » reçoit ce nombre.

Si « chance » est inférieure ou égale à 10, un événement positif se produit. Par exemple, le bonheur du Tamagotchi est augmenté de 20 points, sans dépasser 100. Un message est affiché pour indiquer que le Tamagotchi a trouvé un jouet.

Si « chance » est supérieure ou égale à 90, un événement négatif se produit. La santé du Tamagotchi est réduite de 20 points, mais ne peut pas tomber en dessous de 0. Un message est affiché pour indiquer que le Tamagotchi est malade.

En somme, cette méthode ajoute une dimension aléatoire au jeu, créant des événements imprévus qui peuvent affecter positivement ou négativement le Tamagotchi, rendant ainsi le jeu plus dynamique et imprévisible.

## Conclusion

En résumé, les nouvelles fonctions ajoutées pour le vieillissement, la gestion du temps et les événements aléatoires ont apporté profondeur et réalisme au jeu Tamagotchi. La principale difficulté était d'assurer une gestion fluide des entrées utilisateur et d'intégrer de manière cohérente les événements aléatoires, ce qui a été résolu avec succès. Ces ajouts rendent le jeu plus dynamique et imprévisible.

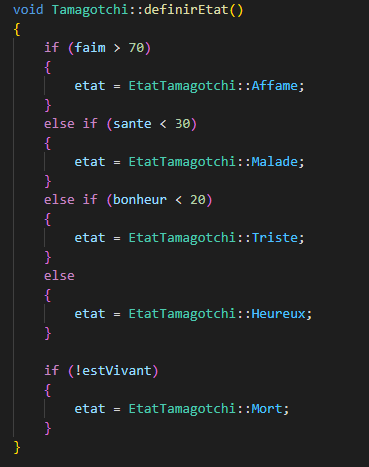
# Étape 7 : Gestion de l'état du Tamagotchi

Définir les états possibles :Avant de commencer, identifiez les différents états que peut prendre votre Tamagotchi en fonction de ses caractéristiques. Par exemple, vous pourriez avoir des états tels que "heureux", "affamé", "malade", "sale", etc.

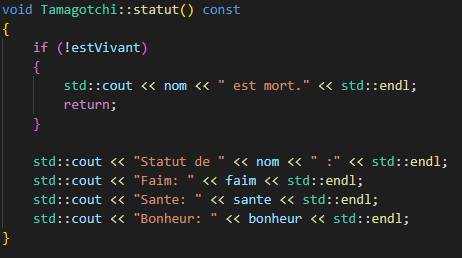
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Mise à jour de l'état : À chaque tour, après que les actions de l'utilisateur ont été effectuées et que le temps a passé, mettez à jour l'état du Tamagotchi en fonction de ses caractéristiques actuelles. Par exemple, si la faim est très élevée et le bonheur est très bas, vous pouvez considérer que le Tamagotchi est "affamé et triste". Adaptez ce code à votre Tamagotchi !



Afficher l'état du Tamagotchi : Affichez l'état actuel du Tamagotchi à l'utilisateur après chaque tour pour qu'il puisse voir comment ses actions et le passage du temps ont affecté l'état du Tamagotchi.

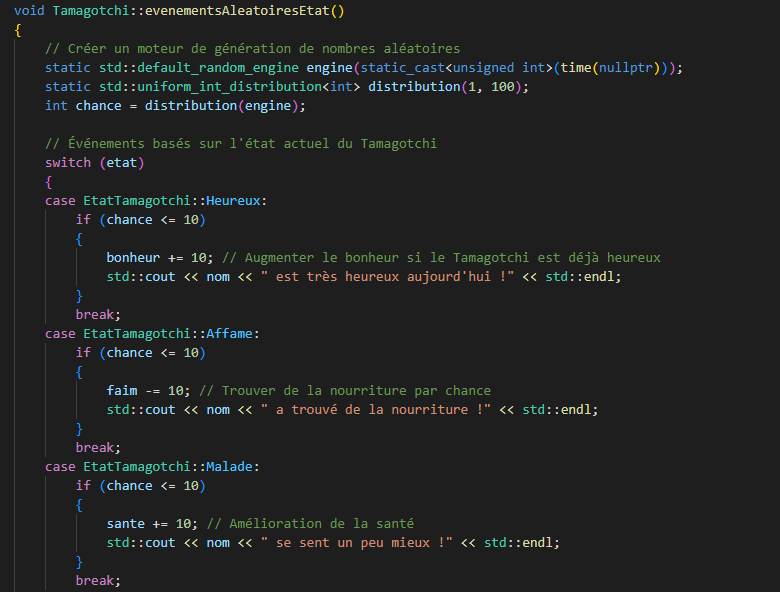


Réagir aux états critiques : Lorsque l'état du Tamagotchi devient critique (par exemple, "affamé et triste" ou "malade"), vous pouvez déclencher des événements spéciaux ou des conséquences. Par exemple, si le Tamagotchi est "malade", vous pouvez réduire davantage sa santé ou lui faire perdre des points de bonheur.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Événements aléatoires basés sur l'état : Vous pouvez également ajouter des événements aléatoires qui se produisent en fonction de l'état actuel du Tamagotchi. Par exemple, si le Tamagotchi est "heureux", il pourrait avoir la chance de trouver un objet spécial qui améliore son bonheur.



# Étape 8 : Gestion de la sauvegarde

Créer une classe de gestion de la sauvegarde : Pour gérer la sauvegarde de l'état du Tamagotchi, créez une classe dédiée à la gestion de la sauvegarde. Cette classe doit avoir des méthodes pour sauvegarder et charger l'état du Tamagotchi.

Implémenter la sauvegarde : Dans la méthode sauvegarderTamagotchi(), vous pouvez utiliser les opérations de fichier pour sauvegarder l'état du Tamagotchi dans un fichier texte ou binaire. Vous pouvez choisir le format de sauvegarde qui vous convient le mieux.

Implémenter le chargement : Dans la méthode chargerTamagotchi(), vous pouvez lire les données sauvegardées à partir du fichier et les utiliser pour recréer l'état du Tamagotchi.