

Une image contenant cercle, capture d’écran, texte, diagramme

Le contenu généré par l’IA peut être incorrect.

15 octobre 2025

Dubel Nathan, 26K

Gymnase de Bienne et du Jura Bernois

Patrick Hochuli, Informatique

Programmation du Bao game

Un jeu traditionnel africain

Table des matières

[1. Introduction au jeu 3](#_Toc211690807)

[1.1 Origine du jeu 3](#_Toc211690808)

[1.2 Explication et règles du jeu 3](#_Toc211690809)

[2. Commencement et déroulement du travail 3](#_Toc211690810)

[2.1 Démarche et logique 3](#_Toc211690811)

[2.2 Le code en quelques lignes 4](#_Toc211690812)

[3. Critique du travail effectué 7](#_Toc211690813)

[3.1 Déroulement du travail 7](#_Toc211690814)

[3.2 Travail complet 8](#_Toc211690815)

[3.3 Chemin parcouru 8](#_Toc211690816)

[4. Sources et bibliographie 9](#_Toc211690817)

[5. Remerciements 9](#_Toc211690818)

# 1. Introduction au jeu

## 1.1 Origine du jeu

Le Bao game, aussi appelé « mancala » , est un jeu de stratégie originaire d’Afrique de l’Est, pratiqué aujourd’hui essentiellement dans les pays africains parlant le swahili. L’origine du nom, pour la plupart inconnue, provient de la traduction directe de « graine » en swahili, soit « bao ». Il existe plusieurs variantes de ce jeu ; ici je me suis inspiré de la variante « kujifunza » qui veut dire littéralement « pour les touristes » en swahili, car c’est la variante qu’on m’a enseigné lors de mon voyage en Tanzanie.

## 1.2 Explication et règles du jeu

Le bao game se joue à deux, les deux joueurs étant des adversaires. Chacun des joueurs dispose de deux rangées de huit trous, chaque trou étant rempli de deux graines, soit trente-deux graines par joueur. Le but du jeu est simple : éliminer toutes les graines de son adversaire. Ainsi, chacun leur tour, les deux joueurs vont jouer un coup qui leur permet d’atteindre ce but.

Un joueur doit choisir un trou, prendre les graines de ce trou et les distribuer une par une dans les trous suivants, dans le sens inverse des aiguilles d’une montre. Si la dernière graine atterrit dans un trou non vide (donc rempli d’une graine minimum), alors ce joueur répète le processus avec les graines se trouvant dans ce trou. Si la dernière graine atterrit dans un trou vide, alors le tour du joueur s’arrête. Ce joueur élimine les graines de son adversaire lorsque le trou vide est situé sur la rangée limitrophe à la rangée de l’adversaire, auquel cas les graines se trouvant dans le trou en face sont éliminées.

Cas spécial : au premier tour, un joueur n’a le droit de choisir qu’un trou situé sur la rangée limitrophe à la rangée de l’adversaire.

# 2. Commencement et déroulement du travail

## 2.1 Démarche et logique

Ma démarche a consisté en trois parties principales :

* Représentation du plateau et des graines
* Déplacement et capture des graines
* Règles du jeu et interface utilisateur

Tout d’abord, je me suis dit qu’il fallait que je code le plateau et le jeu. Une fois cela programmé, il fallait que je m’occupe des graines : d’abord les placer dans le jeu et ensuite réussir à les déplacer en cliquant dessus. J’ai procédé étape par étape : je ne déplaçais qu’une seule graine au début, puis les 32 graines d’un joueur, puis les 64 de tout le jeu, les graines se déplaçant de manière circulaire (il y a deux rangées) dans le sens contraire des aiguilles d’une montre. J’ai ensuite codé la capture de graine et j’en ai profité pour afficher le nombre de graines capturées par les joueurs en dehors du plateau. Il me restait alors une étape à franchir : les règles.

Effectivement, mon code fonctionnait mais le tour des joueurs n’était pas respecté, la fin d’une partie n’était pas programmée et le cas spécial du premier tour n’était pas pris en compte non plus. Alors, il a fallu que je code des messages et que je bloque le jeu en cas de non-respect des règles. Ainsi, avec l’aide de mon mentor, j’ai pu terminer mon jeu. Cependant, en jouant au jeu pour tester mon code, j’ai réalisé que les graines se déplaçaient effectivement mais uniquement la position finale était affichée. Cela était problématique car un joueur qui désigne un trou a alors du mal à suivre les déplacements et ne peut se rendre compte d’une erreur de calcul. J’ai voulu donc coder les animations des graines et j’ai réalisé que cela revenait à modifier mon code entier, et le recommencer. J’ai alors décidé de coder une pause, à chaque déplacement, afin que le joueur puisse suivre quand même.

Une fois que le jeu fonctionnait, j’ai amélioré l’interface visuel en modifiant la taille des graines pour qu’on puisse mieux les voir, en rajoutant des couleurs pour les deux adversaires (rouge et bleu) et en affichant le numéro des rangées pour mieux visualiser. Finalement, j’ai affiché un message à chaque tour indiquant que c’est au tour de l’autre de jouer, pour éviter toute confusion sur la personne qui doit effectivement jouer.

## 2.2 Le code en quelques lignes

J’ai codé mon jeu en langage Python sur un fichier .py que je pouvais modifier sur l’application Visual Studio Code et j’ai enregistré ce fichier sur Github.com, pour y donner accès à mon mentor.

Afin d’optimiser l’explication du code (qui représente plus de 400 lignes), je vais me concentrer uniquement sur la partieUne image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect. des déplacements de graines, les trous et les captures. J’ai représenté mon jeu par une matrice, dans laquelle chaque trou correspond à une case de la matrice et chaque graine correspond au contenu de ces cases, donc à la valeur des cases de la matrice. J’ai utilisé le code ci-dessous qui permet de créer cette matrice.

Ainsi, pour désigner une case de la matrice, il suffit d’utiliser « jeu[x][y] », x étant la ligne et y la colonne souhaitée.

Ensuite, pour qu’une graine se déplace lorsque je clique dessus, j’ai créé un « event » que j’ai relié avec le clic de souris grâce au code « bind() ».

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Puis, la question qui se posait c’était comment relier les cordonnées d’un clic au numéro de case correspondant ? J’ai codé de manière à ce que chaque trou soit distancé d’exactement 100 pixels. Ainsi, il me suffisait de diviser par cent les cordonnées d’un clic pour obtenir le numéro de ligne(li) et le numéro de colonne(co) correspondant.

Comme ça, « jeu[li][co] » correspond exactement au contenu du trou sur lequel on clique. C’est beaucoup plus simple, par exemple, pour déplacer les graines d’un trou appartenant à la rangée du dessus (point de vue du joueur 1), il suffit d’ajouter un à la valeur des trous suivants, qui sont donc les mêmes cordonnées que le trou de départ, mais avec un numéro de colonne(co) différent et ce, jusqu’au début de la rangée.

Les graines se déplaçant dans le sens inverse des aiguilles d’une montre, le trou après « jeu[li][co] » est tout simplement « jeu[li][co-1] » et le trou suivant sera « jeu[li][co-1-1] », etc. jusqu’au début de la rangée. Donc, le nombre de trous après le trou initial correspond au nombre de graines se trouvant dans ce trou, les graines étant distribuées une par une dans les trous suivants. Et si le nombre de graines est plus grand que le nombre de trous qui restent jusqu’au début de la rangée, alors il faut distribuer les graines restantes dans les trous de la rangée du bas (toujours du point de vue du joueur 1). Et inversement, si le trou initial est sur la rangée du bas, il faudra distribuer les graines une par une dans les trous correspondants à « jeu[li][co+nombre de graines] ».

Peu importe notre point de vue sur la partie (que ce soit du joueur 1 ou du joueur 2), toutes les graines du jeu se déplacent dans la même direction, dans le sens inverse des aiguilles d’une montre. Le code étant écrit du point de vue du joueur 1, le code pour le joueur 2 est alors exactement le même, en termes de déplacement ! Donc ce qui fonctionne pour le joueur 1, fonctionne exactement de la même manière pour le joueur 2. Ceci a grandement facilité la logique de codage.

Voici un cas de figure où le nombre de graines d’un trou se situant sur la rangée du haut est plus petit que le nombre de trous jusqu’au début de la rangée :

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

Le contenu généré par l’IA peut être incorrect.

Ici, j’ai utilisé k qui correspond au nombre de graines d’un trou et + « . » car ce sont des graines et pas des nombres que j’affiche. Donc les graines d’un trou ne sont pas un nombre, mais une suite de caractères (ici : une graine = .). Pour transformer cette chaîne de caractère en nombre et pouvoir modifier la valeur des trous suivants par rapport à ce nombre, j’ai utilisé la fonction len(), qui donne le nombre de caractères présents dans une chaîne.

Pour modifier la valeur d’une case de matrice, j’ai utilisé le code « itemconfig », qui change la valeur d’un objet désigné.



Ainsi, la valeur d’une case de matrice est modifiée, mise à jour et affichée en temps réel grâce au lien entre la matrice et le jeu que j’ai codé juste avant.

Aussi, à la fin d’un déplacement, il me suffisait d’écrire que le dernier trou devient alors le premier trou, comme si on avait cliqué dessus. Le code est alors lancé en boucle, jusqu’à ce que la dernière graine atterrisse dans un trou vide. J’ai alors ajouté ce code à la fin de mon « event » :

Une image contenant texte, capture d’écran, Police

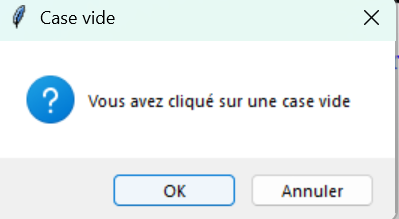
Le contenu généré par l’IA peut être incorrect.

Ce code stipule que si la graine atterrit dans un trou vide (donc que la valeur du dernier trou = 1), et que ce trou est situé sur une rangée limitrophe aux deux joueurs, on capture les graines qui sont dans le trou d’en face, comme le disent les règles du jeu. Les graines capturées sont alors enregistrées sur le compteur de chaque joueur et lorsqu’un joueur arrive à capturer les 32 graines de son adversaire, la partie se termine. Aussi, peu importe la position du dernier trou sur telle ou telle rangée (li == 0,1,2 ou 3), le code affiche un message expliquant que c’est au tour de l’autre de jouer et sort donc de la condition (break) et recommence le code tout au début, mais pour le joueur suivant.

Finalement, il ne me restait plus qu’à coder les messages qui apparaissent lorsqu’on joue un coup illégal et pour cela, j’ai voulu développer au maximum l’interface et l’interaction du jeu avec les joueurs en utilisant des « messagebox askokcancel ». J’ai codé ainsi : une certaine variable a une certaine valeur. Si cette variable a la valeur attribuée, alors on entre dans la condition. Dans la condition, si le clic est effectué sur une mauvaise rangée, alors un message apparaît. Sinon, une nouvelle valeur est attribuée à la variable, correspondant à une autre condition. Dans cette autre condition, le principe est le même, mais elle est adaptée pour l’autre joueur. Au début de la partie, le principe est le même mais dans les coups illégaux, Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.on y trouve aussi la rangée non limitrophe au joueur adverse.

Ainsi, à chaque tour, si le coup est illégal, un message explicatif apparaît et si le coup est légal, le prochain joueur joue. J’ai aussi rajouté un code qui fait apparaître un message lorsqu’on clique sur une case vide, pour éviter toute confusion. A la fin, un message de fin de partie termine effectivement la partie en affichant le vainqueur. Pour cela, j’ai fait en sorte de faire apparaitre un message lorsque le nombre de graines capturées atteint les 32.

**  
*Exemple de messagebox utilisé*

# 3. Critique du travail effectué

## 3.1 Déroulement du travail

Je n’ai eu aucun mal à commencer mon travail de maturité. Pour moi, ce que j’avais à faire était clair et je me suis tout de suite mis au travail. Cependant, quand j’ai eu mon deuxième rendez-vous avec mon mentor, M. Hochuli, j’ai vite compris que ce n’était pas aussi simple que je ne le pensais. Effectivement, j’avais été très ambitieux, je voulais, en plus du code, programmer une intelligence artificielle. Je pensais que j’aurai eu le temps de tout faire mais j’ai oublié un détail important : je n’avais jamais codé, j’étais sans expérience.

Mon mentor m’a affirmé qu’il fallait d’abord que je programme le jeu en tant que tel avant de m’attaquer à l’IA, et je l’ai donc écouté. Au début, malgré ma motivation, les mécanismes de programmation étaient encore flous. En effet, ce que je programmais ne fonctionnait presque jamais et je devais souvent recommencer. Alors, après quelques rendez-vous avec mon mentor, il m’a conseillé de coder pas à pas, étape par étape afin d’avancer au lieu de recommencer. J’ai donc poursuivi mon travail de maturité avec ce raisonnement-là. J’avançais bien, jusqu’à mon premier vrai obstacle : le déplacement de plusieurs graines. N’ayant aucune expérience, je n’avais aucune idée de comment faire et donc mon premier réflexe a été d’aller de nouveau demander à mon mentor. Il m’a expliqué la fonction de code des matrices et je suis resté bloqué pendant quelques temps car je ne comprenais pas le concept. Après mûre réflexion, j’ai enfin eu un déclic, ce qui m’a permis d’avancer !

Le mécanisme s’est répété plusieurs fois, avec le déplacement « circulaire » des graines, la capture des graines, les tours des joueurs à respecter et finalement, les animations. Les animations ont été la seule fois où je n’ai pas trouvé de déclic, alors j’ai pensé à cette alternative de mettre des pauses entre chaque déplacement.

J’ai dû travailler pendant la plupart des vacances, les vacances étant les seuls moments que j’avais à disposition pour bien me poser et avancer. La fin était un peu plus intense encore, avec les tests écrits réguliers. J’en tire les conclusions suivantes : la nécessité d’anticiper et de s’organiser, ne jamais abandonner et toujours chercher des solutions. C’est un raisonnement qui m’a particulièrement aidé lors de ce travail et qui m’accompagnera pour la suite.

## 3.2 Travail complet

Ce qui a bien fonctionné dans ce travail, c’était surtout la motivation et le plaisir que je prenais à coder, mélangeant logique et projection, en particulier lorsqu’on voit le résultat visuel de ce qu’on a programmé.

En revanche, ce qui a moins bien fonctionné, c’est la mise en place d’un travail régulier. En effet, prévoir ce que j’allais coder et planifier une structure de travail à l’avance n’était pas facile, car la plupart du temps je devais recommencer une partie du code. Un autre point aussi, c’est la difficulté de trouver des sources qui permettent de traduire une logique dans notre tête avec un langage python. La plupart du temps, j’étais obligé de demander à mon mentor comment faire et d’attendre sa réponse lors de nos rendez-vous.

Avec du recul, si je devais refaire ce travail de maturité, j’aurais dès le début réfléchi à une structure du code plus robuste et je me serais informé auprès de plus de sources, afin d’être moins dépendant des aléas de programmation et des feedbacks de mon mentor. J’aurais pu ainsi mieux concentrer mon énergie et mes efforts de manière continue et efficace tout au long du projet.

## 3.3 Chemin parcouru

En commençant ce projet, je n’avais aucune expérience en code. Certes, j’avais quelques bases des cours facultatifs de robotique de la deuxième année, mais, n’étant pas en OC informatique, je n’avais aucune compétence pour programmer un code comme celui-ci. Cependant, au fur et à mesure que le projet avançait, j’apprenais à coder et ce, de manière de plus en plus complexe. J’ai commencé à coder le visuel du jeu, puis à coder les déplacements, puis les relations et la logistique des graines, en complexifiant le code à chaque fois. Pour moi, ce projet était instructif et m’a permis d’acquérir des connaissances spécifiques en langage Python comme les messagebox ou l’apparition d’une interface visuelle, tout au long du projet, un peu comme en OC informatique. Il m’a aussi permis de mieux cerner les implications d’un travail autonome, comme déjà énoncé.

Pour conclure, je suis très content et fier de mes réalisations et des connaissances acquises et je pense que je pourrai maintenant coder d’autres jeux ou fonctions de manière beaucoup plus structurée.

# 4. Sources et bibliographie

Pour ce projet, je n’ai pas beaucoup de sources. En effet, la plupart du travail étant de la logique, je dirais que les sources venaient principalement de mes raisonnements. Cependant, avant de commencer, j’ai lu et pris connaissance du document que mon mentor avait mis sur Teams, un document explicatif des différents codes de base en Python. Au début, je m’appuyais sur ce document mais plus mon travail avançait, plus j’avais besoin de codes python complémentaires. Alors j’effectuais quelques recherches, sans jamais vraiment trouver ce qu’il me fallait. Je demandais donc presque toujours à mon mentor la suite ; il était donc une de mes sources principales.

Parfois, j’utilisais ChatGPT. Non pas pour écrire du code à ma place, mais pour connaitre le les fonctions nécessaires qui n’étaient pas marqué dans le document ou alors comprendre les raisons d’un crash.

<https://chatgpt.com/share/68e924a2-c720-8000-8791-e8f9b957aee2>

<https://chatgpt.com/share/68ea2106-a21c-8000-a847-9797bf0809b9>

<https://chatgpt.com/share/68ea211b-6a38-8000-a499-75c6b716203e>

<https://chatgpt.com/share/68ea212f-5dd0-8000-9bab-7977c9a898fa>

Voici la seule vidéo tutoriel que j’ai regardé, au tout début, pour comprendre la mécanique du « bind » : <https://www.youtube.com/watch?v=Jb5Df2ul41M>

# 5. Remerciements

Un grand merci à mon mentor qui m’a soutenu et aidé jusqu’à la fin dans mon travail, et qui m’a mis dans la bonne direction dès le début. Durant ce projet, j’avais énormément de questions et de doutes et mon mentor a su répondre présent à chaque fois.

Merci à mes parents et mes amis, qui ont su m’aider lorsque je ne comprenais pas ce qui ne marchait pas, et qui m’ont permis de garder ma motivation lorsque je la perdais parfois.