

TRAVAIL NOTÉ 2

Développement d'un jeu vidéo simple (40%)

Feuille d'identité

Nom Moya Prénom Melissa

Numéro d'étudiant 24332062 Trimestre Automne 2025

Nom de la personne responsable de l'encadrement miloud.mihoubi@telug.ca

Date 2025-11-17

Réservé à l'usage de la personne responsable de l'encadrement

Date de réception _____ Date de retour _____

Note _____

Résumé

Ce rapport présente un jeu vidéo 2D développé en Python, intitulé Chasse Express. Le jeu utilise la bibliothèque Pygame pour les animations 2D, intègre une gestion des erreurs par exceptions, et suit les principes de la programmation fonctionnelle. Ce document décrit la conception du jeu, son fonctionnement, la gestion des erreurs, et l'approche fonctionnelle adoptée. Une vidéo explicative est incluse pour démontrer le jeu en action. Les sources consultées, notamment la documentation Pygame et l'assistance fournie par l'IA GitHub Copilot, sont référencées à la fin du rapport.

Description du jeu

Chasse Express est un jeu vidéo en deux dimensions développé en Python à l'aide de la bibliothèque Pygame. Il s'inscrit dans le genre adresse et tir et s'inspire du classique Duck Hunt tout en proposant une approche renouvelée. L'objectif est de combiner la simplicité d'interaction avec un dynamisme visuel et des éléments originaux favorisant l'immersion.

Le joueur incarne un chasseur accompagné d'un chien Sheltie. La partie débute par une interaction avec le chien qui déclenche un aboiement et fait apparaître des pies à capturer avant la fin du temps imparti. Deux contraintes structurent la mécanique de jeu, soit des munitions limitées et un chronomètre, ce qui introduit une dimension stratégique et impose précision et optimisation. L'action repose sur l'utilisation de la souris pour viser et tirer. La difficulté évolue progressivement par l'augmentation de la vitesse des cibles, la réduction du temps disponible et la limitation des munitions, ce qui renforce l'engagement et la rejouabilité.

Sur le plan esthétique, le jeu présente des graphismes originaux inspirés des paysages de l'Alberta. Les personnages adoptent un style cartoon tandis que les décors privilégient un réalisme visuel. Les sprites personnalisés et les animations fluides contribuent à la cohérence graphique. L'ambiance sonore immersive soutient l'atmosphère générale et améliore l'expérience utilisateur.

Choix du concept

Le concept de Chasse Express s'inscrit dans une démarche pédagogique visant à illustrer de manière concrète les notions fondamentales du cours de programmation Python par la réalisation d'un jeu d'adresse inspiré du classique Duck Hunt. Ce projet met en œuvre des techniques modernes telles que l'animation en deux dimensions, l'interactivité par la souris et la gestion multimédia à l'aide de la bibliothèque Pygame.

La conception graphique repose sur deux approches complémentaires. D'une part, le dessin dynamique des pies illustre l'utilisation des fonctions graphiques et des primitives de dessin. D'autre part, l'intégration de ressources externes telles que des sprites et des images démontre la gestion des fichiers multimédias et leur exploitation dans un environnement interactif.

Ce projet mobilise un ensemble diversifié de compétences en programmation. Il inclut la gestion des événements utilisateur tels que les clics de souris et les actions du jeu, la gestion du temps et des ressources avec un chronomètre et des munitions limitées, ainsi que la programmation orientée objet pour la modélisation des entités animées. Les structures de contrôle, notamment les boucles et les conditions, assurent la logique et l'animation. Le contrôle du framerate est réalisé grâce à la fonction pygame.time.Clock afin de garantir une fluidité optimale. La robustesse est renforcée par la gestion des erreurs via des blocs try-except et la documentation systématique facilite la maintenance et l'évolution du code.

Chasse Express constitue ainsi une application cohérente des principes du génie logiciel tels que l'encapsulation, la modularité, la gestion rigoureuse des exceptions et la documentation. Ce projet concrétise l'application des concepts théoriques abordés en cours et démontre leur intégration dans une solution interactive et fonctionnelle.

Gestion des dépendances

Pour ce projet, l'outil pip a été retenu pour la gestion des dépendances Python en raison de sa simplicité, de sa fiabilité et de sa large adoption dans la communauté. Il permet l'installation et la maintenance des bibliothèques nécessaires, notamment Pygame, via la commande pip install pygame.

Dans le dépôt du projet, un fichier requirements.txt a été inclus afin d'assurer la portabilité et la reproductibilité de l'environnement. Ce fichier liste l'ensemble des dépendances nécessaires, conformément aux bonnes pratiques du génie logiciel. Il permet de recréer l'environnement rapidement grâce à la commande pip install -r requirements.txt.

Bien que des gestionnaires plus récents tels que uv proposent des fonctionnalités avancées, pip demeure la solution la plus appropriée pour un projet pédagogique en raison de sa compatibilité et de son support étendu. Son association avec des environnements virtuels (venv) garantit l'isolation des bibliothèques et prévient les conflits, conformément aux recommandations du cours.

Gestion des erreurs par exceptions

Dans Chasse Express, la gestion des erreurs par exceptions est centrale pour garantir la stabilité et la convivialité du programme.

1. Gestion de l'importation de Pygame

Le premier point sensible est l'importation de la bibliothèque Pygame, indispensable au fonctionnement du jeu. Si Pygame n'est pas installé sur la machine de l'utilisateur, une erreur d'importation (ImportError) se produira. Pour éviter qu'une telle erreur ne génère une trace confuse et un plantage du programme, le code place l'import dans un bloc try-except. Si l'exception est levée, un message explicite est affiché, invitant l'utilisateur à installer la dépendance manquante, puis le programme s'arrête proprement.

```
try:  
    import pygame  
except ImportError:  
    print("Pygame est requis. Installez-le avec : pip install pygame")  
    sys.exit(1)
```

chasse_express.py (lignes 13 à 18)

Cette gestion préventive améliore la convivialité du jeu en orientant directement l'utilisateur vers la solution, évitant ainsi une interruption brutale et incompréhensible.

2. Initialisation de Pygame

Une fois la bibliothèque importée, l'initialisation du moteur graphique et audio constitue une autre étape critique. Des erreurs peuvent survenir, par exemple si les modules ne se chargent pas correctement en raison d'un problème système ou d'une incompatibilité. Le code protège cette étape par un bloc try-except global capturant toute exception (Exception), ce qui garantit que le jeu ne démarre pas dans un état instable.

```
try:
    pygame.init()
    pygame.mixer.init()
except Exception as e:
    print(f"Erreur lors de l'initialisation de Pygame : {e}")
    sys.exit(1)
```

chasse_express.py (lignes 54 à 59)

Cette approche assure que toute anomalie lors de l'initialisation est détectée et traitée immédiatement, évitant des comportements imprévisibles par la suite.

3. Création de la fenêtre de jeu

La création de la fenêtre graphique est une opération sensible, dépendante du matériel et des pilotes graphiques. Une erreur à ce stade (pygame.error) peut empêcher l'affichage du jeu. Le code place donc la création de la fenêtre dans un bloc try-except dédié, capturant ces erreurs spécifiques et affichant un message explicite avant de quitter proprement.

```
try:
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption('Chasse Express')
except pygame.error as e:
    print(f"Erreur lors de la création de la fenêtre : {e}")
    sys.exit(1)
```

chasse_express.py (lignes 64 à 69)

Cette gestion ciblée permet d'anticiper les problèmes liés à la configuration graphique et d'informer l'utilisateur de manière claire.

4. Gestion des erreurs de ressources

Un autre point critique du jeu concerne le chargement des ressources indispensables, telles que les images et les sons. Si une ressource essentielle est absente ou corrompue, le jeu ne peut pas fonctionner correctement. Pour gérer ce cas, une exception personnalisée `ErreurRessourceJeu` est définie. Lorsqu'une ressource ne peut être chargée, cette exception est levée, ce qui permet d'arrêter le programme proprement et d'afficher un message explicite à l'utilisateur.

```
class ErreurRessourceJeu(Exception):
    """
    Exception levée lors d'une erreur de chargement de ressource du jeu.
    """
    pass
```

resources.py (lignes 93 à 97)

Cette gestion centralisée des erreurs de ressources améliore la robustesse du jeu et facilite le diagnostic en cas de problème, en évitant des plantages imprévus et des messages d'erreur peu explicites.

5. Gestion des erreurs de configuration

Lors de la sélection de la difficulté, des erreurs de type `KeyError`, `ValueError` ou `TypeError` peuvent survenir si la configuration est incorrecte ou incomplète. Le code capture ces exceptions pour revenir proprement au menu principal sans planter le jeu.

```
except (KeyError, ValueError, TypeError) as e:
    print(f"Erreur de configuration de difficulté : {e}")
    menu = True
    continue
```

chasse_express.py (lignes 332 à 335)

6. Gestion des erreurs inattendues dans les boucles de jeu et de menu

Des blocs `try-except` génériques capturent toute exception inattendue dans la boucle du menu ou du jeu, ce qui évite un crash brutal et permet d'afficher un message d'erreur pour faciliter le débogage.

Conception fonctionnelle

La programmation fonctionnelle est intégrée dans Chasse Express à travers l'utilisation de fonctions pures, de listes en compréhension, d'expressions génératrices et de principes d'immuabilité, ce qui favorise la clarté, la modularité et la robustesse du code.

Fonctions pures

De nombreuses fonctions utilitaires du projet sont des fonctions pures, car elles prennent des arguments, retournent un résultat et n'ont pas d'effets secondaires sur l'état global. Par exemple, dans `utils.py`, les fonctions `calcule_score`, `consomme_munition`, `verifie_victoire`, et `verifie_defaite` sont toutes pures :

```
def calcule_score(score, increment=1):
    return score + increment

def consomme_munition(ammo, amount=1):
    return max(0, ammo - amount)
```

utils.py (lignes 21 à 27)

Ces fonctions garantissent une prévisibilité et facilitent les tests unitaires.

Listes en compréhension et expressions génératrices

Le projet utilise des listes en compréhension pour générer dynamiquement des collections d'objets, comme dans la création des entités pies :

```
magpies = [Magpie.create_random(speed, HEIGHT, MAGPIE_BODY_RADIUS) for _ in
range(magpie_count)]
```

chasse_express.py (ligne 225)

Des expressions génératrices sont également utilisées pour des calculs efficaces et lisibles, par exemple pour le calcul des couleurs dégradées et de la largeur totale du titre :

```
color = tuple(int(c1[j] + (c2[j] - c1[j]) * t) for j in range(3))
```

chasse_express.py (ligne 158)

```
total_width = sum(title_font.size(ch)[0] for ch in title_text)
```

chasse_express.py (ligne 160)

Immuabilité et initialisation fonctionnelle

L'immuabilité est encouragée par l'utilisation de lambdas dans les dataclasses pour initialiser les listes sans effet de bord :

```
pos: List[float] = field(default_factory=lambda: [0.0, 0.0])
vel: List[float] = field(default_factory=lambda: [0.0, 0.0])
```

entities.py (lignes 19 à 20)

En synthèse, les fonctions utilitaires du projet évitent de modifier directement l'état global ou les objets passés en argument, ce qui limite les effets de bord et facilite la maintenance. Ainsi, ces pratiques fonctionnelles viennent compléter l'approche orientée objet, en apportant modularité et prévisibilité.

Code source principal

La structure du projet Chasse Express repose sur une organisation modulaire, chaque fichier ayant une responsabilité précise :

Figure 1 – Organisation modulaire du projet Chasse Express

```
Chasse Express/
|
├── main.py          # Lance le jeu, appelle le module principal
├── chasse_express.py # Logique principale du jeu, boucle de jeu, gestion des ressources
├── entities.py      # Définition des entités animées (chien, pies, etc.)
├── ui.py            # Fonctions d'affichage et interface graphique
├── resources.py     # Chargement et gestion des ressources (images, sons, polices)
├── settings.py      # Constantes et paramètres de configuration
├── utils.py          # Fonctions utilitaires diverses
|
└── assets/           # Ressources multimédias (images, sons)
    ├── images/
    └── audio/
|
└── tests/            # Tests unitaires et de validation
```

*Diagramme généré par GitHub Copilot.

Ce diagramme illustre la décomposition du projet en modules spécialisés, favorisant la clarté, la réutilisabilité et la maintenance du code.

Introduction au module principal

Le module principal de Chasse Express constitue le point d'entrée du jeu. Il orchestre l'initialisation de l'environnement graphique et sonore, le chargement des ressources, la gestion du menu

principal, ainsi que le déroulement de la boucle de jeu. Ce module centralise la logique de lancement et de coordination des différentes composantes du projet, assurant ainsi une organisation claire et modulaire du code.

Extraits clés du module principal (`chasse_express.py`)

1. Initialisation de Pygame et des ressources

Cette partie du code initialise les modules graphiques et audio de Pygame, crée la fenêtre de jeu, puis charge les ressources essentielles (images, sons). Chaque étape est protégée par des blocs `try/except` pour garantir que le jeu ne démarre que si tout est prêt.

```
try:
    pygame.init()
    pygame.mixer.init()
except Exception as e:
    print(f"Erreur lors de l'initialisation de Pygame : {e}")
    sys.exit(1)

try:
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption('Chasse Express')
except pygame.error as e:
    print(f"Erreur lors de la création de la fenêtre : {e}")
    sys.exit(1)

try:
    sheltie_img = load_image(SHELTIE_IMG_PATH)
    if sheltie_img:
        sheltie_img = pygame.transform.smoothscale(sheltie_img, (200, 170))
    else:
        raise ErreurRessourceJeu(f"Image non trouvée : {SHELTIE_IMG_PATH}")
    # ... chargement des autres ressources ...
except ErreurRessourceJeu as e:
    print(f"Erreur critique de ressource : {e}")
    sys.exit(1)
```

chasse_express.py (lignes 51 à 100)

Le jeu commence par l'initialisation de Pygame et de son module audio. Si une erreur survient, un message explicite est affiché et le programme s'arrête proprement. Ensuite, la fenêtre de jeu est créée. Si le matériel ou les pilotes ne sont pas compatibles, une erreur est capturée et le jeu s'arrête également. Enfin, les ressources graphiques et sonores sont chargées et si une ressource essentielle est absente, une exception personnalisée est levée pour informer l'utilisateur et arrêter le jeu. Cette gestion centralisée garantit que le jeu ne démarre jamais dans un état instable.

2. Boucle principale du jeu et gestion des événements

La boucle principale du jeu gère l'affichage, les interactions utilisateur et la logique de progression.

Elle orchestre le déroulement du jeu en fonction des actions de l'utilisateur.

```

while running:
    if menu:
        try:
            # Affichage du menu et gestion des boutons
            # ...
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
                    # ... gestion du choix de la difficulté ...
                    pygame.display.flip()
                    clock.tick(60)
        except Exception as e:
            print(f"Erreur dans le menu : {e}")
            continue

    # ... autre partie du code : vérification de la difficulté ...

    try:
        settings = DIFFICULTY_SETTINGS[difficulty]
        # ... initialisation de la manche ...
        while running_round:
            # Affichage, gestion du chien, des pies, du viseur, du panneau
            # d'état
            # ...
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                    running_round = False
                elif event.type == pygame.MOUSEBUTTONDOWN:
                    # ... gestion des clics sur le chien, les pies, etc ...
                    pygame.display.flip()
                    clock.tick(60)
        except Exception as e:
            print(f"Erreur dans la boucle de manche : {e}")
            running_round = False
    
```

chasse_express.py (lignes 125 à 335)

La boucle principale commence par afficher le menu et attend que l'utilisateur choisisse une difficulté. Ensuite, elle lance la partie, gère l'affichage des éléments (chien, pies, viseur, etc.) et traite les événements utilisateur (clics, fermeture de la fenêtre). Les événements sont traités pour déclencher les actions appropriées (démarrer la partie, tirer sur une pie, revenir au menu, etc.). En

cas d'erreur inattendue pendant la partie, une exception est capturée pour éviter un crash brutal et permettre un retour propre au menu.

3. Exemple de gestion d'une entité (extrait de entities.py)

Les entités du jeu, comme la pie, sont modélisées par des classes orientées objet. Voici un extrait de la classe Magpie, qui gère la position, le déplacement et la détection des collisions.

```
@dataclass
class Magpie:
    pos: List[float] = field(default_factory=lambda: [0.0, 0.0])
    vel: List[float] = field(default_factory=lambda: [0.0, 0.0])
    flying_away: bool = False
    fly_away_timer: int = 0

    @classmethod
    def create_random(cls, speed: float, screen_height: int, body_radius: int) -> "Magpie":
        start_x = body_radius
        start_y = random.randint(body_radius, screen_height - 150 - body_radius)
        vx = speed * random.uniform(0.9, 1.2)
        vy = random.uniform(-2, 2)
        return cls(pos=[start_x, start_y], vel=[vx, vy])

    def update(self, speed: float, width: int, height: int, body_radius: int) -> None:
        # ... mise à jour de la position et gestion des rebonds ...
```

entities.py (lignes 14 à 35)

La classe Magpie représente une pie dans le jeu. Elle possède des attributs pour la position, la vitesse et l'état (en fuite ou non). La méthode create_random permet de générer une pie à une position et avec une vitesse aléatoires. La méthode update gère le déplacement de la pie et les rebonds sur les bords de l'écran. Cette structure orientée objet facilite la gestion et l'évolution des entités du jeu.

Le code complet est disponible dans l'archive ZIP jointe au devoir ainsi que sur mon profil GitHub à l'adresse suivante : [moyamelissa/Chasse_Express](https://github.com/moyamelissa/Chasse_Express)

Vidéo explicative

Une vidéo démontrant le jeu est incluse dans l'archive zip sous le nom Demo_Chasse_Express.mp4. Elle présente le déroulement d'une partie complète pour chaque

niveau offert soit facile, moyen, difficile. Alternativement, la vidéo est accessible sur YouTube via [Demo Chasse Express](#).

Discussion et analyse

Cette analyse présente les principaux défis rencontrés, les choix techniques effectués ainsi que les méthodes de validation mises en œuvre lors du développement de Chasse Express. Elle souligne les enseignements tirés et la démarche adoptée pour assurer la qualité et la robustesse du projet.

Défis rencontrés

La conception de Chasse Express a été à la fois stimulante et enrichissante. J'ai pris plaisir à réaliser ce projet, qui m'a permis d'approfondir ma compréhension de GitHub et de présenter mon travail de façon professionnelle, comme on le fait dans le domaine des TI. J'ai découvert la création d'un dépôt, l'ajout de documentation (README avec badges) et l'utilisation des GitHub Actions.

Un défi particulier a été d'adopter une nomenclature en français. Par habitude, j'utilise l'anglais en programmation, mais j'ai privilégié le français pour tout ce qui concerne l'expérience du joueur comme les titres, les menus et les communications. J'ai également rédigé les commentaires du code et les messages d'exception en français afin de respecter la langue du cours. En revanche, j'ai conservé l'anglais pour la logique interne et la structure du code afin d'assurer la réutilisabilité du projet dans mon portfolio. Ce choix me permet de présenter un travail compréhensible par des recruteurs et développeurs anglophones tout en démontrant ma capacité à travailler en français.

Sur le plan technique, la structuration du code en modules clairs et réutilisables a joué un rôle essentiel. Le développement a débuté dans un fichier unique, puis une phase de refactorisation a permis de découper la logique du jeu en plusieurs composants afin d'éviter un fichier principal trop volumineux. Cette approche a favorisé la lisibilité, la maintenabilité et la cohérence globale du projet. J'ai aussi choisi de ne pas complexifier le jeu. Initialement, mon objectif était de reproduire un jeu de type Qubix, mais j'ai rapidement constaté la complexité mathématique et algorithmique que cela impliquait. J'ai donc opté pour un concept plus simple afin de me concentrer sur l'exploration approfondie des notions abordées en cours.

Enfin, j'ai pris soin d'éliminer le code mort ou non utilisé, ce qui a nécessité une relecture attentive et des ajustements réguliers afin de garantir la qualité et la robustesse du projet.

Tests et validation

Tout au long du développement, j'ai mis l'accent sur la qualité du code grâce à différents tests automatisés. J'ai utilisé les frameworks unittest et pytest pour vérifier le bon fonctionnement des fonctions et modules clés du projet. La couverture des tests a été mesurée afin de m'assurer que les fonctionnalités critiques étaient bien vérifiées.

L'intégration de workflows GitHub Actions a permis d'automatiser l'exécution des tests à chaque modification du code, garantissant ainsi la stabilité continue du projet. Grâce à cette initiative, le dépôt affiche un badge CI dans le README, attestant que toutes les actions et tous les tests passent avec succès. Cela renforce la crédibilité et la fiabilité du projet auprès des utilisateurs et des contributeurs.

Figure 2 – Badge CI dans le fichier README.md

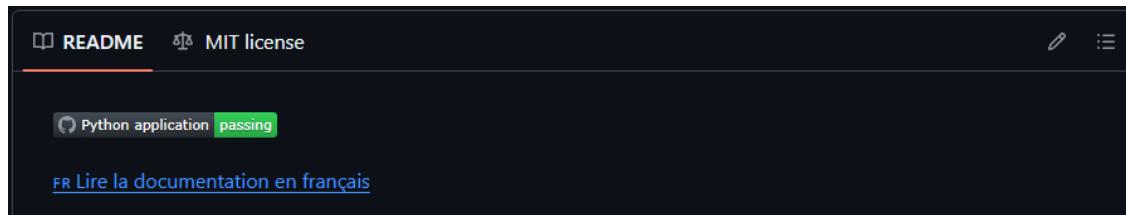


Figure 3 – Résultats de test et du rapport de couverture des tests

```
pygame 2.6.1 (SDL 2.28.4, Python 3.13.1)
Hello from the pygame community. https://www.pygame.org/contribute.html
-----
Ran 33 tests in 0.221s

OK
Name           Stmts  Miss  Cover
----- 
chasse_express.py      224   215    4%
entities.py            82     3   96%
resources.py           66     30   55%
settings.py             21     0  100%
tests\test_chasse_express.py  170     1   99%
ui.py                  107    39   64%
utils.py                20      8   60%
-----
TOTAL                 690   296   57%
```

Le rapport montre que 33 tests unitaires ont été exécutés avec succès en 0,221 seconde, attestant de la stabilité du projet. La couverture globale du code est de 57%. Bien que certains modules comme chasse_express.py (4%) et resources.py (55%) présentent des marges d'amélioration, les modules critiques tels que settings.py (100%) et entities.py (96%) sont bien couverts.

Concepts appris

Au fil du projet, j'ai pu mettre en pratique et approfondir plusieurs concepts clés de la programmation Python, notamment la structuration modulaire du code, l'utilisation des classes et des dataclasses, la gestion des exceptions et la manipulation de bibliothèques externes comme Pygame. J'ai également découvert l'importance des tests unitaires et de la couverture de code pour garantir la fiabilité et la robustesse d'une application. Ce projet, que je pensais initialement très difficile, m'a permis de constater que la complexité diminue lorsque la documentation est claire et que la structure du code est bien définie. Une fois ces fondations établies, le développement devient essentiellement une question d'écriture de code.

L'intégration de GitHub et de ses outils collaboratifs m'a permis de mieux comprendre le cycle de vie d'un projet logiciel et de valoriser mon travail en le partageant dans un contexte professionnel. Cette démarche répondait à mon objectif de créer un projet réutilisable dans mon portfolio, ce qui m'a motivé à me familiariser avec les fonctionnalités clés de GitHub telles que la gestion de versions, la documentation, l'automatisation des tests et la validation continue.

Au-delà des aspects techniques, ce projet m'a sensibilisé à l'importance de valoriser l'expérience utilisateur. Une documentation claire du code, un partage structuré sur GitHub et la création d'une vidéo explicative ont été des éléments essentiels pour rendre le projet compréhensible et attractif. Ces efforts visent à faciliter la réutilisation du code par d'autres développeurs et à démontrer la valeur d'un projet bien présenté. Partager la vidéo avec des amis et collègues a confirmé cette approche, car leurs réactions positives ont montré que la présentation donnait envie d'essayer le jeu. Tout au long du développement, je me suis interrogé sur les choix liés à l'interface utilisateur, notamment l'impact des couleurs, des polices, des icônes, des images et même des défis sur l'expérience globale du joueur. Cette réflexion m'a permis d'intégrer des principes de design simples, mais efficaces pour améliorer l'ergonomie et l'attrait visuel du jeu.

Enfin, le choix d'une nomenclature en français pour l'expérience utilisateur m'a sensibilisé à l'importance de la clarté et de l'accessibilité du code, notamment dans un contexte académique ou collaboratif. Ce compromis entre français pour l'interface et anglais pour la logique interne illustre la nécessité d'adapter le code à différents publics tout en respectant les exigences pédagogiques.

Dans le futur, j'aimerais rendre ce projet entièrement bilingue afin de le rendre accessible à un public plus large. Je souhaite également explorer des solutions ergonomiques adaptées aux personnes ayant des limitations visuelles ou auditives, telles que des options pour les daltoniens, des contrastes renforcés, des polices lisibles, des icônes explicites et des alternatives sonores ou visuelles pour les joueurs sourds. Ces améliorations visent à promouvoir l'inclusivité et à offrir une expérience utilisateur optimale pour tous.

En conclusion, ce projet m'a permis de consolider mes acquis en programmation Python, en gestion de projet et en utilisation des outils collaboratifs modernes, tout en explorant avec plaisir la création d'un jeu vidéo. J'ai expérimenté l'ensemble du cycle de développement, de la conception à la validation, en passant par la documentation et le partage sur GitHub. Cette démarche est, à mon avis, une expérience extrêmement formatrice et valorisante!

IA générative

Il est important de noter que la plupart des réponses générées par l'IA ont été appliquées directement dans mon code, souvent sous forme de correctifs ou d'exemples à adapter. Cependant, j'ai systématiquement validé, testé et adapté ces suggestions pour garantir leur cohérence avec l'architecture globale du projet et les bonnes pratiques enseignées dans le cours. Cela rend difficile la sauvegarde ou la documentation précise de chaque échange, car l'IA a servi principalement d'assistant technique pour débloquer des situations concrètes, plutôt que de fournir des réponses théoriques ou des explications à conserver. Mon utilisation de l'IA s'est donc inscrite dans une démarche pratique et intégrée au processus de développement. Ci-dessous est un aperçu des types de questions que j'ai posées et des réponses courtes afin de respecter la longueur demandée du devoir. L'utilisation de Github pour mon code est aussi déclarée dans mon Readme, afin de tester que le développement de mon code a été réalisé avec support Github copilot, un système qui je crois est incroyablement puissant et un support incroyable pour une étudiante comme moi qui réalise pour la première fois un jeu vidéo et supporte l'écriture du code.

Documentation des types de requêtes à l'IA générative

1. Assurance qualité et vérification de code

- Q : Peux-tu vérifier s'il y a du code mort ou non utilisé dans mon projet ?
➤ R : J'ai analysé ton code. Aucune fonction majeure inutilisée détectée.

- Q : Peux-tu réviser la qualité du français de mes commentaires dans mon code ?
➤ R : Voici les corrections principales :
 - dessiner_arbre : "surface donner" → "surface donnée"
 - entities.py ligne 8 : Ajout de "et état" après "vitesse"
 - resources.py ligne 15 : Précise "et la met en cache"

Globalement tes commentaires sont clairs. Pense à ajouter des docstrings complètes avec Args>Returns pour les fonctions principales.

➤ Q : Est-ce que mes tests couvrent bien toutes les fonctionnalités critiques du projet ?

➤ R : Couverture actuelle : 57%.

Bien couvert : settings.py (100%) et entities.py (96%)

Problème critique : chasse_express.py seulement 4% couvert

Priorités : Ajoute des tests d'intégration simulant une partie complète, teste les cas d'erreur (fichiers manquants, init Pygame échoué), et vise 70%+ global.

➤ Q : Comment puis-je améliorer la structure de mes modules pour plus de clarté?

➤ R : Il serait pertinent de regrouper les fonctions utilitaires dans un module dédié, de séparer clairement la logique du jeu et l'interface graphique, et de documenter chaque module avec un en-tête descriptif.

➤ Q : À partir de combien de lignes un fichier .py devient-il trop long et devrait-il être restructuré en modules ?

➤ R : Il n'existe pas de règle stricte, mais en pratique :

- Au-delà de 200 à 300 lignes, un fichier Python commence souvent à devenir difficile à maintenir et à lire.
- À partir de 400 lignes, il est fortement recommandé de le découper en plusieurs modules/fichiers selon les responsabilités (logique métier, entités, interface, utilitaires, etc.).

L'essentiel est :

- De regrouper les fonctions/classes par thème ou fonctionnalité.
- De faciliter la navigation, la réutilisation et les tests.

Un bon indicateur : Si tu dois souvent faire défiler ou chercher dans le fichier, ou si tu as du mal à retrouver une fonction, il est temps de le restructurer.

2. Conseils ergonomiques et expérience utilisateur

- Q : Peux-tu suggérer des couleurs qui rendraient l'interface plus agréable ?
- R : Utilise des couleurs douces et contrastées, par exemple un fond bleu clair, des boutons verts pour les actions positives, et des touches d'orange pour attirer l'attention.

- Q : Comment rendre le menu principal plus intuitif ?
- R : Ajoute des icônes aux boutons, agrandis les zones cliquables, et propose un retour visuel (changement de couleur) au survol de la souris.

3. Dépannage technique

- Q : J'ai une erreur "image not found" alors que le fichier est bien dans le dossier, que faire ?
- R : Vérifie le chemin d'accès à l'image et privilégie l'utilisation de chemins absous avec os.path.join pour éviter les erreurs liées au répertoire courant.

- Q : Comment corriger un bug où le jeu ne se ferme pas proprement ?
- R : Assure-toi d'appeler pygame.quit() et sys.exit() à la fin de la boucle principale pour garantir la fermeture correcte du programme.

4. Optimisation et bonnes pratiques

- Q : Comment puis-je améliorer la performance de l'animation dans Pygame ?
- R : Utilise pygame.time.Clock() pour contrôler le framerate et évite de recharger les images ou sons à chaque frame : charge-les une fois au début et réutilise-les.

5. Rédaction du rapport

- Q : Peux-tu me faire un diagramme du plan de mon code pour mon rapport (structure des modules et des relations) ?
- R : [Voir image]

Références

Antidote. (2025). « *Antidote, un logiciel de correction et d'aide à la rédaction développé par Druide Informatique, a été utilisé pour la révision linguistique et stylistique du rapport.* ».

GitHub. (2025). *YouTube - GitHub Channel*. Retrieved from YouTube:
<https://www.youtube.com/@GitHub>

GitHub Copilot :. (2025). « GitHub Copilot, un outil d'assistance à la programmation basé sur l'IA développé par GitHub et OpenAI, a été utilisé pour générer des suggestions de code et améliorer la structure du projet. ».

Godin, R., & Lemire, D. (2014). *Programmation avec Python : des jeux au Web*.

pygame. (2025). *pygame*. Retrieved from pygame documentation:
<https://www.pygame.org/docs/>