

Licence MIASHS parcours MIAGE deuxième année

## Rapport de projet Algorithme et Programmation



Projet réalisé du 4 novembre 2025 au 6 janvier 2026

**Enseignant référent :** M. Delbot

**Auteurs :** 42013418 Rakotoarivelo Shelly-Linda  
43012479 Ly Abdoulaye

**Code source complet :**  
<https://github.com/Abdoulaye-collab/PROJET-L2>

## **Remerciements**

Nous tenons tout d'abord à remercier notre professeur, M. DELBOT, ainsi que l'ensemble de l'équipe pédagogique pour la qualité de leur enseignement et leur encadrement tout au long de ce semestre. Ce projet a été l'occasion de mettre en pratique les concepts théoriques vus en cours et de nous confronter aux réalités du développement logiciel.

Nous les remercions également de nous avoir laissé une grande liberté dans le choix du sujet, ce qui nous a permis de laisser libre cours à notre créativité et de travailler sur un projet qui nous tenait à cœur.

Enfin, nous remercions nos camarades pour l'entraide et les échanges constructifs, ainsi que toutes les personnes qui ont pris le temps de tester le jeu et de nous faire des retours précieux lors des phases de débogage.

# Table des matières

<b>Remerciements</b>	<b>2</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Contexte du projet . . . . .	5
1.2 Genèse du projet : Le défi technique . . . . .	5
1.3 Présentation de "Wizard Battleship" . . . . .	5
<b>2 Environnement de travail et Organisation</b>	<b>6</b>
2.1 Outils et Méthodes collaboratives . . . . .	6
2.2 Organisation : Une approche itérative . . . . .	6
<b>3 Méthodologie : L'IA comme Assistant de Développement</b>	<b>7</b>
3.1 Apports Principaux . . . . .	7
3.2 Limites et Regard Critique . . . . .	7
<b>4 Description du projet et Game Design</b>	<b>8</b>
4.1 Concept et Règles . . . . .	8
4.2 Système de "Loot" (Gestion de Main) . . . . .	8
4.3 L'Arsenal (Les Cartes Sortilèges) . . . . .	8
4.4 Identité Visuelle, Sonore et Narrative . . . . .	9
<b>5 Réalisation Technique et Développement</b>	<b>11</b>
5.1 Organisation des Fichiers . . . . .	11
5.2 Flux d'exécution et Gestion des Entrées . . . . .	12
5.3 Algorithmes de Rendu et Optimisation . . . . .	12
5.4 Mathématiques Appliquées (Physique et Animation) . . . . .	13
5.5 Intelligence Artificielle : Architecture Hybride . . . . .	14
5.6 Robustesse du Code (Duck Typing) . . . . .	15
5.7 Cycle de Développement et Itérations . . . . .	15
<b>6 Bibliothèques et Technologies</b>	<b>17</b>
6.1 Langage et Environnement . . . . .	17
6.2 Bibliothèques Externes . . . . .	17
6.3 Modules Standards Python . . . . .	17
<b>7 Difficultés rencontrées</b>	<b>18</b>
7.1 Choix du moteur graphique : De Arcade à Pygame . . . . .	18
7.2 Synchronisation Modèle-Vue . . . . .	18
7.3 Incohérence d'affichage des résultats (Débogage) . . . . .	18
7.4 La logique des Cartes Spéciales . . . . .	18
<b>8 Bilan</b>	<b>19</b>
8.1 Bilan du projet . . . . .	19
8.2 Pistes d'Amélioration (Perspectives) . . . . .	19
<b>9 Annexes</b>	<b>21</b>
9.1 Guide d'Installation et d'Exécution . . . . .	21
9.2 Configuration de l'IA (Prompt Engineering) . . . . .	22
9.3 Constantes d'Équilibrage (settings.py) . . . . .	22

9.4	Crédits et Ressources . . . . .	23
9.5	Galerie : Parcours d'une Session de Jeu . . . . .	24

# 1 Introduction

## 1.1 Contexte du projet

Ce projet a été réalisé dans le cadre de l'unité d'enseignement "Algorithmique et Programmation" en deuxième année de Licence (L2). Le cahier des charges initial était ouvert : nous devions concevoir et développer un projet informatique complet, avec une forte incitation à intégrer des outils d'Intelligence Artificielle (IA) générative, soit dans le processus de développement, soit au cœur du gameplay.

## 1.2 Genèse du projet : Le défi technique

Le choix de notre sujet a suivi un cheminement itératif. Initialement, nous nous sommes orientés vers une **Bataille Navale** traditionnelle. Ce choix nous semblait constituer une base solide et maîtrisable. Cependant, lors de la validation du sujet, notre enseignant nous a mis en garde : la mécanique classique de ce jeu est souvent jugée trop simple techniquement pour un niveau universitaire L2, à moins d'y implémenter une IA complexe basée sur un LLM (Large Language Model) pour gérer l'adversaire.

Face à cette contrainte, nous avons mené une analyse de risques. Tout miser sur une IA connectée (type GPT) comportait des dangers majeurs : latence du réseau, réponses incohérentes ou "hallucinations" du modèle qui auraient pu rendre le jeu injouable.

Nous avons alors transformé cette contrainte en opportunité via une approche hybride :

- **Techniquement** : Nous avons sécurisé le projet en connectant le jeu à une vraie IA (via l'API Hugging Face) tout en implémentant un algorithme de secours robuste (fallback) pour garantir la stabilité.
- **Ludiquement** : Pour dépasser la simplicité initiale, nous avons totalement repensé le Game Design autour du thème "Sorcier". Ce choix n'est pas uniquement esthétique ; il légitime l'introduction de mécaniques complexes (gestion de mana, cartes de sorts, effets de zone) qui seraient incohérentes dans une bataille navale militaire classique.

## 1.3 Présentation de "Wizard Battleship"

Le résultat de ce travail est "**Wizard Battleship**" (L'Héritage des Arcanes). Bien plus qu'une simple reproduction du jeu de société, c'est une expérience immersive qui plonge le joueur dans la peau d'un sorcier des mers. L'objectif n'est pas seulement de détruire la flotte adverse le plus rapidement possible, mais de le faire avec sang-froid. Le joueur doit gérer ses ressources (cartes sortilèges) et prendre des décisions pondérées : faut-il utiliser une "Bombe" maintenant ou la garder pour une "Salve" dévastatrice plus tard ? C'est cet équilibre entre rapidité et réflexion qui définit l'identité de notre jeu.

## 2 Environnement de travail et Organisation

### 2.1 Outils et Méthodes collaboratives

Pour structurer notre collaboration à distance, nous avons établi un écosystème d'outils complémentaires :

- **Développement** : Visual Studio Code a été notre IDE principal, configuré avec les linters Python.
- **Planification** : Un document de suivi partagé (Google Docs) servait de feuille de route dynamique pour lister les tâches et suivre l'avancement.
- **Versionning (Git/GitHub)** : Travail sur un dépôt commun. Cela nous a permis de fusionner le code du moteur (Backend) géré par Abdoulaye et celui de l'interface (Frontend) par Shelly, tout en gérant les conflits. Nous avons adopté une stratégie de branches. La branche `main` était sanctuarisée pour le code stable. Le développement des nouvelles fonctionnalités (interface, effets) s'effectuait sur une branche dédiée (*feature branch*), permettant de tester les modifications sans risquer de compromettre le projet principal.
- **Communication** : Discord a servi de bureau virtuel pour les réunions de conception et le partage d'écran lors des sessions de débogage.

### 2.2 Organisation : Une approche itérative

Notre méthode de travail a évolué en deux grandes phases constructives :

1. **Phase 1 (Le Squelette)** : Abdoulaye a d'abord mis en place l'architecture technique du projet (logique fondamentale, connexion IA, grille). À ce stade, nous avions un prototype fonctionnel mais visuellement austère.
2. **Phase 2 (L'Expérience)** : Shelly a ensuite transformé le prototype en un jeu complet. Elle a retravaillé l'interface, ajouté la narration en 3 phases (Bureau/- Grimoire/Combat) et intégré les effets visuels.

Domaine	[SHELLY] (Front & UX)	[ABDOULAYE] (Back & IA)
Architecture & IA	Intégration graphique & Mode Target	<b>Responsable</b> (Squelette, API HuggingFace)
Interface (UI)	<b>Responsable</b> (Fenêtres, Grilles, Menu, Orbe)	Support technique
Gameplay Visuel	<b>Responsable</b> (Particules, Feedbacks)	<b>Responsable</b> (Logique Cartes, Audio)
Gestion de Projet	<b>Responsable</b> Documentation (Rapport/README), Gestion des conflits	<b>Initialisation</b> (Git), Gestion des conflits

TABLE 1 – Matrice de répartition des tâches

### 3 Méthodologie : L'IA comme Assistant de Développement

Au-delà de l'IA intégrée au gameplay (l'adversaire "Léviathan"), nous avons utilisé les modèles génératifs (principalement ChatGPT et Gemini) comme assistants de développement ("Co-pilotes") tout au long du projet. Cette approche nous a permis de gagner du temps sur les tâches répétitives pour nous concentrer sur la logique complexe et l'architecture.

#### 3.1 Apports Principaux

L'aide de l'IA a été décisive sur trois aspects majeurs :

- **Génération de "Boilerplate"** : Au début du projet, l'IA nous a aidés à générer rapidement les structures de base des classes `Player` et `Grid`, nous évitant le syndrome de la page blanche.
- **Refactoring et Modularisation** : C'est sans doute l'apport le plus critique. Lorsque le fichier principal est devenu trop massif (+600 lignes), l'IA a assisté le découpage du code en modules distincts (`game.py`, `utils.py`, `settings.py`). Cette étape minutieuse a été essentielle pour garantir la lisibilité et la maintenabilité du projet.
- **Résolution de problèmes techniques** :
  - *Débogage* : Analyse rapide des traces d'erreurs (Stack Traces) pour identifier les causes de crashes.
  - *Mathématiques* : Suggestion de formules, comme l'utilisation de `math.sin()` pour l'animation fluide de l'orbe dans le menu.
  - *Regex* : Aide à la construction d'expressions régulières pour nettoyer proprement le JSON renvoyé par l'API Hugging Face.

#### 3.2 Limites et Regard Critique

Si l'IA est un accélérateur, elle s'est aussi révélée faillible. Nous avons identifié plusieurs cas où une intervention humaine stricte était indispensable :

**Hallucinations Techniques** À plusieurs reprises, les modèles nous ont proposé d'utiliser des méthodes de `pygame` qui n'existent pas (ou plus). *Exemple* : L'invention d'une fonction native pour dessiner des rectangles arrondis, alors qu'il fallait en réalité implémenter une fonction personnalisée.

**Perte de Contexte** Sur les fichiers longs, l'IA avait tendance à "oublier" les modifications récentes ou les variables définies dans d'autres fichiers. Nous avons dû apprendre à ne lui fournir que les contextes pertinents plutôt que de lui demander de gérer l'intégralité du code.

**Conclusion** : L'IA ne nous a pas remplacés. Elle a agi comme un catalyseur, mais la décision finale sur l'architecture et la logique est toujours restée entre nos mains.

## 4 Description du projet et Game Design

### 4.1 Concept et Règles

Notre jeu, *Wizard Battleship*, revisite la bataille navale classique en transposant l'affrontement militaire dans un duel de sorciers.

- **Base classique** : Une grille de 10x10 et une flotte de 5 navires à positionner secrètement.
- **Condition de victoire** : La destruction totale de la flotte adverse (ou mode "Mort Subite" envisagé en amélioration).
- **Innovation** : Le remplacement des missiles par des sortilèges et l'ajout d'une gestion de ressources (cartes).

### 4.2 Système de "Loot" (Gestion de Main)

Contrairement aux jeux utilisant une jauge de mana, nous avons opté pour une mécanique de "deck-building" simplifié.

- **Probabilité et Loot** : Le jeu distribue au joueur une "main" de 3 cartes tirées aléatoirement parmi 6 sorts disponibles. Cette limitation introduit une probabilité (et donc une rareté) qui force le joueur à s'adapter.
- **Consommables** : Chaque carte est à usage unique. Une fois le sort lancé, la carte disparaît de l'inventaire.
- **Sécurisation (UX)** : Nous avons soigné l'ergonomie. Le joueur peut cliquer sur une carte pour l'activer ("Armer le sort"), mais il peut également la désélectionner en recliquant dessus s'il change d'avis. Cela évite de gâcher un bonus précieux sur une erreur de clic.

### 4.3 L'Arsenal (Les Cartes Sortilèges)

Voici les effets implémentés pour enrichir la stratégie :

#### 1. Les Cartes Offensives

- **Salve** : L'attaque ultime. Elle déclenche un barrage d'artillerie sur toute la ligne ciblée (colonne 0 à 9), maximisant les chances de toucher plusieurs navires d'un coup.
- **Bombe** : Lance une frappe aérienne à des coordonnées aléatoires. Le joueur ne choisit pas la cible, c'est le hasard qui décide où l'explosion aura lieu sur la grille ennemie.
- **Double Tir** : Octroie immédiatement une munition supplémentaire, permettant de jouer deux fois de suite.

#### 2. Les Cartes Tactiques

- **Radar** : Permet d'espionner une case précise. Le jeu révèle si elle contient un navire ou de l'eau, sans tirer.
- **Sabotage** : Une attaque informatique sur l'IA. Elle annule le tour de l'adversaire, offrant un tour gratuit au joueur.
- **Bouclier** : Active une protection temporaire sur la flotte du joueur pour encaisser les futurs tirs ennemis.

## 4.4 Identité Visuelle, Sonore et Narrative

L'expérience utilisateur a été pensée comme un voyage immersif en quatre actes. Nous avons voulu que le joueur ne soit pas simplement face à un logiciel, mais qu'il incarne un sorcier. Chaque interface possède une identité propre, soutenue par des arrière-plans générés par IA et un design sonore spécifique.

### 1. L'Accueil : Le Calme avant la Tempête

- L'écran titre installe le contexte narratif.
- **Immersion** : Un arrière-plan généré par IA représentant des sorciers naviguant sur une mer agitée accueille le joueur.
  - **Ambiance** : Des orbes magiques volent librement sur l'écran pour dynamiser l'image statique. Les titres introduisent la dualité visuelle du jeu : **Cyan** pour les forces du joueur, **Violet** pour l'ennemi.

### 2. La Cabine du Capitaine (Identification)

La transition vers la saisie du nom se fait dans un cadre intimiste.

- **Décor** : Le joueur est visuellement installé à un bureau, face à une fenêtre donnant sur l'océan. Un grimoire flotte au centre, attendant d'être signé.
- **Interaction (Orbe)** : L'écriture du nom est traitée comme un rituel magique. L'orbe qui remplace la souris suit le tracé des lettres, laissant une traînée lumineuse persistante, simulant l'encre magique.

### 3. Le Grimoire Tactique (Phase de Placement)

Le jeu bascule littéralement "à l'intérieur" du livre.

- **Direction Artistique** : L'interface technique de la grille est camouflée en page de parchemin ancien. La typographie (style "Harry Potter") et les teintes brunes renforcent l'aspect authentique.
- **Mécanique Visuelle** : Placer un bateau équivaut à invoquer un sort. Lorsqu'un navire est positionné depuis la liste de droite, une explosion de particules valide l'action, matérialisant la flotte sur le papier.

### 4. Le Duel des Arcanes (Phase de Combat)

Le grimoire se referme pour laisser place à la réalité du combat en haute mer. C'est l'écran le plus riche en informations (HUD) et en effets.

- **Atmosphère** : L'arrière-plan dévoile l'océan avec des portails magiques lumineux à l'horizon. L'écran s'anime d'énergies néons (Cyan et Violet).
- **Interface de Combat (HUD)** :
  - **Les Grilles** : A droite et à gauche, identifiables par leur code couleur (Cyan pour le joueur, Violet pour l'IA) et leur taille (petite pour le joueur, grande pour l'IA).
  - **L'Inventaire** : Les cartes sortilèges sont disposées en bas à gauche, prêtes à être activées.
  - **Tableau de Chasse** : Un panneau central liste la flotte ennemie. Dès qu'un navire est coulé, son nom est rayé, offrant un suivi clair de la progression.
  - **Dialogue IA** : En bas à droite, une zone de texte permet à l'IA de commenter le combat, rendant l'adversaire plus "vivant".

- **Feedback Visuel (Particules)** : Chaque action déclenche une réponse immédiate. Un tir réussi provoque une explosion de particules colorées, tandis qu'un tir dans l'eau génère une simple croix.
- **Design Sonore (SFX)** : L'immersion passe aussi par l'audio, avec des sons distincts pour chaque événement :
  - *Bruit d'explosion* et jingle lors d'un impact ou d'un navire coulé.
  - *Bruit d'eau (Splash)* lors d'un tir manqué.
  - *Chime magique* lors de l'activation d'une carte spéciale.
  - *Musique de fond* épique pour soutenir la tension du duel.

**5. Le Dénouement (Victoire ou Défaite)** L'écran de fin n'est pas un simple texte, mais une illustration narrative du destin du joueur :

- **En cas de Victoire (Cyan)** : Le sorcier est représenté triomphant, bâton levé et éclairé, dominant une mer dangereuse parsemée de rochers runiques. Un poisson géant nage paisiblement à ses côtés, symbolisant sa maîtrise des éléments.
- **En cas de Défaite (Violet)** : Le ton change radicalement. Le sorcier est montré vaincu dans les abysses, vêtements déchirés et bâton brisé. Le poisson géant, auparavant allié, devient ici une menace terrifiante au premier plan.

Bataille Navale Classique	Wizard Battleship (Notre version)
<b>Gameplay</b> : Hasard pur, tir simple.	<b>Gameplay</b> : Gestion de main (Loot de 3 cartes), stratégie de conservation.
<b>Visuel</b> : Grilles statiques.	<b>Visuel</b> : Orbes flottantes, particules, code couleur Cyan/Violet.
<b>Interface</b> : Fenêtre utilitaire.	<b>Interface</b> : Narration en 4 phases (Bureau → Grimoire → Combat → Fin).
<b>IA</b> : Aléatoire simple.	<b>IA Hybride</b> : LLM (Hugging-Face) + Algorithmes.

TABLE 2 – Synthèse : Comparatif Standard vs Version Enrichie

## 5 Réalisation Technique et Développement

Cette section détaille l'architecture logicielle, les défis algorithmiques et les solutions mathématiques mises en œuvre. Le projet est entièrement codé en **Python 3.11** et s'appuie sur la bibliothèque **Pygame** pour la gestion bas niveau de la boucle événementielle.

### 5.1 Organisation des Fichiers

Le projet est structuré autour d'un fichier principal `main.py` qui orchestre les différents modules. Pour simplifier les imports et l'exécution, l'ensemble des scripts Python se situe à la racine du projet, tandis que les ressources multimédias sont isolées dans un dossier `assets`.

Voici l'organisation logique des fichiers du projet :

```
PROJET-L2/
|
|-- main.py                  # Point d'entrée (Boucle principale)
|-- settings.py              # Configuration (Constantes)
|
|   [LOGIQUE JEU]
|-- game.py                  # Gestionnaire de la partie
|-- player.py                # Classes Joueur et Navire
|-- placement.py             # Gestion du Drag & Drop
|-- cards.py                 # Effets des cartes bonus
|
|   [INTERFACE & VISUEL]
|-- menu.py                  # Gestion du Menu Principal
|-- input_name.py            # Saisie des pseudos
|-- GameOver.py              # Écran de fin
|-- draw_utils.py            # Fonctions de dessin
|-- effects.py               # Particules et effets visuels
|-- utils.py                 # Transitions et outils
|
|   [INTELLIGENCE & SYSTEME]
|-- ai_llm.py                # Connexion API (IA)
|-- ai_personalities.py      # Dialogues de l'IA
|-- input_handler.py          # Gestion des événements souris
| sound_manager.py           # Gestion audio
|
|-- assets/
    |-- images/               # Sprites (Bateaux, Fond, UI)
    |-- sounds/               # Bruitages (Tirs, Explosions)
```

## 5.2 Flux d'exécution et Gestion des Entrées

Le jeu ne suit pas un script linéaire mais réagit selon des états définis gérés dans la boucle principale. Cela permet de passer fluidement du menu au jeu sans recharger le programme.

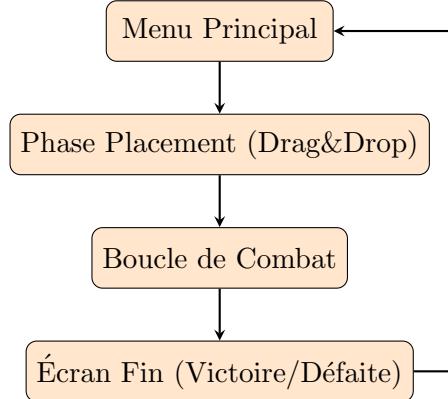


FIGURE 1 – Diagramme simplifié de la machine à états

**Découplage des Contrôles (Input Handler)** Pour respecter le **Principe de Responsabilité Unique (SRP)**, la logique des clics (souris/clavier) a été extraite de la boucle de jeu. Le fichier `game.py` délègue l'événement à `input_handler.py`, qui agit comme un routeur (clic sur carte vs clic sur grille).

```
# game.py : Delegation simple pour garder le code propre
def handle_event(self, event):
    handle_game_events(self, event) # Envoi au module specialise
```

## 5.3 Algorithmes de Rendu et Optimisation

**1. Rendu des Navires (Transparence et Continuité)** Pour concilier esthétique et lisibilité, nous avons développé un rendu hybride dans `draw_utils.py` :

- **Forme Organique** : L'algorithme calcule la longueur totale du navire en pixels selon son orientation (Verticale/Horizontale). Il génère une forme unique englobant toutes les coordonnées. Il applique un `border_radius` élevé (15px) pour arrondir les extrémités, donnant l'apparence d'une véritable coque de navire et non d'un assemblage de blocs. Cela permet d'avoir des navires aux formes fluides et cohérentes, peu importe leur taille.
- **Transparence (Alpha Blending)** : Le rendu utilise une couleur semi-transparente. Cela permet de laisser apparaître les lignes de la grille *sous* le navire. Ainsi, le joueur profite d'un design fluide tout en identifiant précisément les coordonnées des cases occupées.

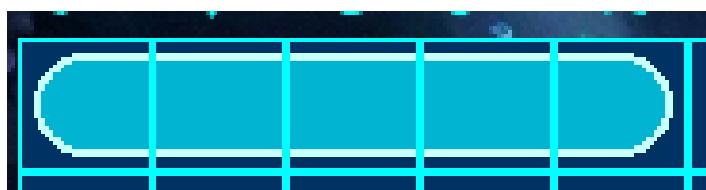


FIGURE 2 – Zoom in-game : La transparence du navire permet de lire la grille en dessous, alliant design courbe et précision tactique.

**2. Optimisation Mémoire (Système de Particules)** La gestion de centaines de particules d'explosion à 60 FPS est coûteuse en ressources. Une erreur classique est de supprimer des éléments d'une liste pendant qu'on la parcourt, causant des bugs d'index.

Dans `effects.py`, nous utilisons une **boucle inversée** (Reverse Loop). En parcourant la liste de la fin vers le début, nous pouvons supprimer (`pop`) les particules "mortes" sans affecter les index des éléments restants.

```
# Parcours inverse pour suppression sécurisée (Garbage Collection)
for i in range(len(self.particles) - 1, -1, -1):
    p = self.particles[i]
    p['timer'] -= 1
    if p['timer'] <= 0:
        self.particles.pop(i)
```

**3. Optimisation Temporelle (Gestion Asynchrone)** Une erreur fréquente avec Pygame est d'utiliser `time.sleep()` pour faire attendre l'IA, ce qui "gèle" la fenêtre (statut "Ne répond pas"). Dans `game.py`, nous avons implémenté un **Timer Non-Bloquant**. Nous décrémentons un compteur (`self.ia_delay`) à chaque frame. Cela permet aux animations (eau, particules) de continuer à tourner fluidement pendant que l'IA "réfléchit".

```
# game.py : Gestion du délai sans figer l'écran
if self.ia_pending and not self.projectile:
    if self.ia_delay > 0:
        self.ia_delay -= 16 # Décompte du temps (1 frame ~ 16ms)
    else:
        self.ai_play() # L'IA joue enfin
        self.ia_pending = False
```

## 5.4 Mathématiques Appliquées (Physique et Animation)

**1. Physique Vectorielle (Projectiles)** Pour l'animation des tirs dans `game.py`, nous n'utilisons pas une interpolation linéaire simple. Nous appliquons la géométrie vectorielle pour garantir une vitesse constante du projectile.

Nous calculons la distance euclidienne (Théorème de Pythagore) entre le tireur et la cible, puis nous normalisons le vecteur de déplacement :

$$Dist = \sqrt{(Target_x - Pos_x)^2 + (Target_y - Pos_y)^2}$$

Ensuite, le déplacement à chaque frame est calculé ainsi :

$$Pos_{new} = Pos_{old} + Vitesse \times \frac{(Target - Pos)}{Dist}$$

**2. Fonctions Sinusoïdales (Pulsation de l'Orbe)** Pour donner vie à l'interface dans `input_name.py`, l'orbe magique possède un effet de "respiration". Au lieu de charger une animation lourde, nous calculons l'opacité (Alpha) en temps réel grâce à une fonction sinusoïdale :

$$\text{Alpha} = |\sin(\frac{\text{Temps}}{300})| \times 200 + 55$$

```
# Extrait input_name.py : Oscillation entre 55 et 255
alpha = int(abs(math.sin(current_time / 300)) * 200) + 55
```

Cette approche mathématique permet une animation infinie, fluide et très légère en mémoire.

## 5.5 Intelligence Artificielle : Architecture Hybride

L'intégration d'une IA générative (LLM) pose des défis de latence et de fiabilité. Une IA purement textuelle pourrait "halluciner" des coordonnées invalides. Nous avons donc conçu, dans `ai_llm.py`, une architecture décisionnelle robuste à **3 niveaux de priorité**.

**Niveau 1 : La Finition (Mode "Target" - Priorité Absolue)** C'est une logique déterministe. Si l'IA a touché un navire au tour précédent, elle "oublie" temporairement toute autre stratégie pour passer en mode chasseur. Elle stocke les coordonnées voisines (Haut, Bas, Gauche, Droite) dans un buffer `targets_buffer` et les vise méthodiquement jusqu'à ce que le navire soit coulé.

```
# Extrait de calculate_ai_move()
# Si on a des cibles prioritaires en memoire
for candidate in targets_buffer[:]:
    r, c = candidate
    # On verifie que la case est jouable (0=Eau, 1=Bateau)
    if player_board[r][c] in [0, 1]:
        return r, c # Tir immediat
```

**Niveau 2 : L'Intuition (Mode LLM - Hugging Face)** Si aucune cible n'est prioritaire (phase de recherche), le jeu sollicite le modèle **GPT-2** via l'API Hugging Face.

- **Construction du Prompt** : L'état du jeu est traduit en texte ("Moves played : A1, B2... Best next move ?").
- **Sécurité (Timeout)** : Pour éviter de figer le jeu, la requête HTTP possède un `timeout` strict de 2 secondes.
- **Parsing** : La réponse JSON est nettoyée via une expression régulière (Regex) pour extraire les coordonnées.

**Niveau 3 : Le Filet de Sécurité (Mode "Checkerboard")** En cas de panne réseau, de latence trop élevée, ou si le LLM renvoie une coordonnée invalide, le jeu bascule automatiquement sur un algorithme probabiliste local : la **Stratégie du Damier**.

Mathématiquement, tout navire de taille  $\geq 2$  occupe forcément au moins une case noire sur un damier. L'IA ne vise donc que les cases dont la somme des coordonnées est paire ( $x + y) \% 2 == 0$ , réduisant l'espace de recherche de 50%.

```
# Fallback : Strategie mathematique du Damier
checkerboard = [p for p in available if (p[0] + p[1]) % 2 == 0]
if checkerboard and len(available) > 40:
    # Optimisation statistique en debut de partie
    row, col = random.choice(checkerboard)
```

**Prompt Engineering et Personnalité** Enfin, pour renforcer l'immersion, le fichier `ai_personalities.py` contient des dictionnaires de dialogues. L'IA adapte ses réactions selon le résultat du tir (Hit/Miss) et le caractère choisi ("Méchante", "Sage", "Gentille"), transformant une simple variable booléenne en une interaction narrative.

(Pour visualiser la structure JSON complète et les prompts utilisés, voir l'Annexe 9.1).

## 5.6 Robustesse du Code (Duck Typing)

Pour éviter les crashes, nous utilisons la flexibilité de Python. Dans `GameOver.py`, l'écran de fin accepte indifféremment un objet `Player` ou une simple chaîne de caractères grâce à l'introspection (`hasattr`) :

```
if hasattr(winner_name, 'name'):
    self.winner_name = winner_name.name # C'est un Objet
else:
    self.winner_name = str(winner_name) # C'est une String
```

## 5.7 Cycle de Développement et Itérations

L'ensemble du projet (Combat, Menu, IA) a suivi une méthodologie de développement itérative. Pour illustrer concrètement cette montée en gamme technique et visuelle, nous prenons ici l'exemple de la **Phase de Placement**, qui a subi les transformations les plus significatives.

**Phase 1 : Le Moteur Logique (V0 - Prototype)** L'objectif initial était de valider les structures de données (matrices dans `player.py`) sans se soucier de l'esthétique.

- **Focus** : Algorithmes de stockage et vérification des coordonnées.
- **Rendu** : Affichage brut de la matrice via des rectangles de couleur (`pygame.draw.rect`). Le placement se faisait par coordonnées textuelles ou clics bruts, sans aide visuelle.

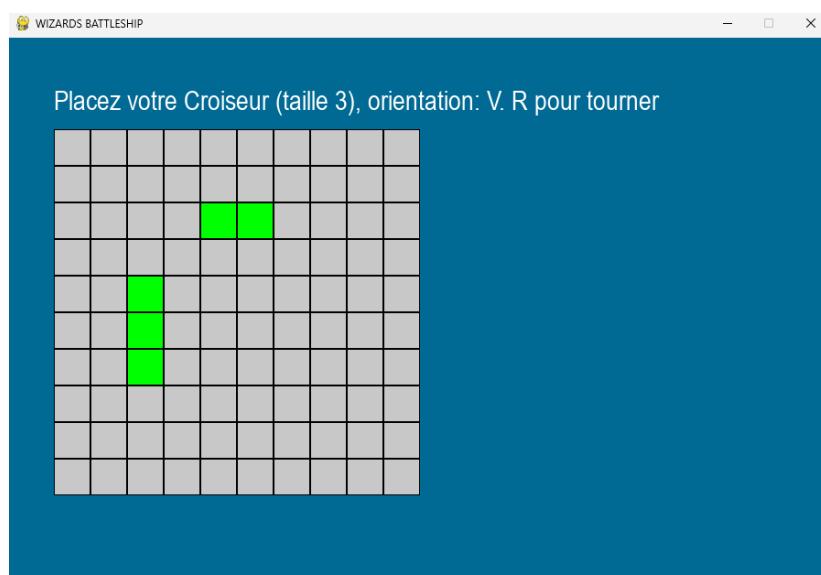


FIGURE 3 – **V0 (Prototype)** : Validation purement technique des règles de la grille.

**Phase 2 : Ergonomie et Interactions (V1 - UX)** Une fois la logique validée, nous avons intégré la couche d’interaction utilisateur (User Experience) dans `placement.py`.

- **Focus** : Implémentation du "Drag & Drop" fluide.
- **Rendu** : Ajout des "Fantômes" (Ghost Ships). Le jeu calcule en temps réel si la position est valide et colore le navire en **Violet** ou **Rouge** avant le relâchement de la souris.



FIGURE 4 – **V1 (Interactivité)** : Le joueur comprend visuellement les règles (violet/-rouge).

**Phase 3 : Immersion et "Game Feel" (V2 - Finale)** La dernière étape a consisté à appliquer l’habillage graphique (Skinning) et les effets spéciaux ("Juiciness").

- **Refonte des Navires** : Abandon des blocs rectangulaires pour des formes continues ("Capsules") avec transparence, permettant une lecture claire de la grille sous le bateau.
- **Atmosphère** : La grille technique est camouflée en page de Grimoire ancien.
- **Feedback** : L’ajout du module `effects.py` déclenche des particules magiques lors de la pose d’un navire, validant l’action par un effet visuel gratifiant.



FIGURE 5 – **V2 (Finale)** : L’interface technique s’efface au profit de l’univers magique.

(Pour visualiser la version finale entière : 9.4).

## 6 Bibliothèques et Technologies

Dans le cadre de notre projet, nous avons sélectionné un ensemble de technologies éprouvées pour garantir la stabilité et la fluidité du jeu.

### 6.1 Langage et Environnement

Le projet est développé en **Python 3.11**<sup>1</sup>. Cette version spécifique a été choisie pour sa parfaite compatibilité avec les modules graphiques récents, tout en offrant des performances optimales pour la boucle de jeu.

### 6.2 Bibliothèques Externes

- **Pygame (Moteur du jeu)**<sup>2</sup> :

C'est la colonne vertébrale du projet. Ce module gère l'affichage des fenêtres, la boucle événementielle (clavier/souris), les sprites (navires, grilles) ainsi que le mixage sonore. Nous l'avons privilégié pour sa robustesse face à la gestion temps réel.

- **Hugging Face (Intelligence Artificielle)**<sup>3</sup> :

Pour animer l'adversaire, nous utilisons l'API d'inférence de Hugging Face. Via le module standard `requests`, le jeu envoie le contexte de la partie à un modèle de langage (LLM) distant qui génère des réponses textuelles dynamiques et contextuelles.

- **Arcade (Testé et abandonné)**<sup>4</sup> :

Initialement envisagée pour sa modernité, cette bibliothèque a été écartée après la phase de prototypage en raison de la complexité de gestion des animations frame par frame (détails en section 6).

### 6.3 Modules Standards Python

En complément, nous exploitons les bibliothèques natives pour la logique interne :

- `sys` : Indispensable pour interagir avec l'interpréteur, il permet notamment de fermer proprement le programme et la fenêtre Pygame lors de la sortie.
- `re` (Regular Expressions) : Utilisé pour le traitement avancé des chaînes de caractères, notamment pour nettoyer ou valider les entrées textuelles et les réponses générées par l'IA
- `random` : Pour la génération aléatoire du placement des navires ennemis et le tirage des cartes.
- `math` : Pour les calculs de trajectoires et les animations sinusoïdales (l'orbe du menu).
- `time` : Pour gérer les délais et le rythme de jeu de l'IA.

---

1. <https://www.python.org/downloads/release/python-3110/>

2. <https://he-arc.github.io/livre-python/pygame/index.html>

3. <https://huggingface.co/docs/inference-providers/tasks/index>

4. <https://api.arcade.academy/en/stable/>

## 7 Difficultés rencontrées

### 7.1 Choix du moteur graphique : De Arcade à Pygame

Initialement, nous avions opté pour la bibliothèque `arcade`, réputée pour sa modernité. Cependant, après une phase de prototypage, nous avons rencontré des limitations bloquantes. Bien que puissante, `arcade` impose une structure rigide qui complexifiait inutilement la gestion de nos animations personnalisées (particules, mouvements de projectiles). De plus, l'affichage d'éléments simples nécessitait un travail de composition fastidieux inadapté à notre besoin. Face à ces contraintes, nous avons pris la décision de migrer vers **Pygame**. Ce changement nous a offert un contrôle plus direct sur la boucle événementielle et a grandement facilité l'intégration des assets graphiques et sonores, stabilisant ainsi le développement.

### 7.2 Synchronisation Modèle-Vue

Une difficulté majeure a résidé dans la synchronisation entre la logique interne (la classe `Player`) et l'affichage. Le jeu doit gérer de nombreuses données simultanément : positions des bateaux, points de vie, cartes en main. Au début, l'actualisation de ces données entraînait des désynchronisations graphiques (problèmes de menus, de tirs fantômes). Nous avons dû restructurer le code pour que chaque modification de l'état du joueur (Logique) soit immédiatement et correctement reflétée à l'écran (Vue).

### 7.3 Incohérence d'affichage des résultats (Débogage)

Une difficulté particulière est survenue lors de la finalisation de l'interface de "Game Over". Dans le panneau récapitulatif des scores, le jeu affichait le nom du perdant à la fois dans la case "Vainqueur" et dans la case "Perdant", créant une confusion totale sur l'issue de la partie.

Le problème était vicieux car le programme ne renvoyait aucune erreur. **Méthode de résolution :** Pour comprendre ce comportement, nous avons dû mettre en place un débogage par traces (logs) dans le terminal. En affichant via des `print` les valeurs des variables `winner` et `loser` juste avant l'affichage graphique, nous avons réalisé que la variable du vainqueur était correctement identifiée par le moteur, mais mal transmise à la fonction d'affichage textuel (une erreur d'inattention dans l'appel des variables). L'utilisation de la console a été décisive pour isoler ce bug invisible.

### 7.4 La logique des Cartes Spéciales

L'implémentation du système de cartes a constitué notre principal défi algorithmique, en particulier pour la carte "**Double Tir**". Cette carte modifie la règle fondamentale de l'alternance des tours. Initialement, le jeu passait automatiquement au tour adverse après l'utilisation de la carte, rendant son effet nul. Le moteur de jeu considérait l'action "utiliser une carte" comme la fin du tour. Après correction du premier bug, un problème de comptage est apparu. Bien que le joueur dispose visuellement de deux tirs, le jeu n'en enregistrait qu'un seul dans la logique interne.

**Solution :** Pour résoudre cette difficulté, nous avons modifié la variable gérant le nombre de tirs autorisés par joueur. Lors de l'activation de la carte, cette variable passe explicitement à 2. Le changement de tour est désormais conditionné par l'épuisement de ce compteur, ce qui a permis de résoudre définitivement le problème.

## 8 Bilan

### 8.1 Bilan du projet

Ce projet nous a permis de mieux comprendre les différentes facettes du développement d'un jeu, et de constater qu'un concept en apparence simple, comme la bataille navale, peut rapidement devenir complexe et stimulant.

Cette complexité repose notamment sur la combinaison de plusieurs défis techniques :

- **Une gestion graphique avancée** (animations, interface) ;
- Une idée originale conçue par nos soins ;
- L'intégration d'un **LLM (Intelligence Artificielle)** au sein de la logique du jeu ;
- **Un système de cartes** venant enrichir le gameplay stratégique.

Ce travail nous a également permis de comprendre le "squelette" d'un jeu vidéo, depuis la logique principale (Main Loop) jusqu'à la gestion de ses mécaniques avancées. Enfin, nous avons acquis une meilleure maîtrise de la **gestion des sons**, de leur synchronisation et de leur impact crucial sur l'expérience de jeu.

### 8.2 Pistes d'Amélioration (Perspectives)

Si le temps le permettait, voici les fonctionnalités identifiées dans notre roadmap pour enrichir l'expérience :

**Interface et Expérience Utilisateur (UX)** Pour rendre le jeu plus confortable et immersif :

- **Menu d'Options** : Ajouter une interface pour régler les volumes sonores (Musique vs Bruitages) et la résolution de la fenêtre en temps réel.
- **Système d'Avatars Réactifs** : Intégrer des portraits ("Mascottes") qui réagissent émotionnellement selon l'état de la partie (sourire si le tir touche, grimace si le joueur est touché).
- **Personnalisation** : Laisser le choix de l'allégeance au début (Team Cyan ou Team Violet), modifiant ainsi tout le thème graphique.

**Nouveaux Modes de Jeu** Pour augmenter la rejouabilité :

- **Mode "Mort Subite"** : Une variante hardcore où chaque joueur ne possède qu'un seul navire. La moindre erreur est fatale.
- **IA Modulable** : Un sélecteur de difficulté à l'entrée :
  - *Novice* : Tirs purement aléatoires.
  - *Stratège* : L'IA actuelle (Chasse + LLM).
  - *Omniscient* : Une IA "Impossible" qui connaît l'emplacement des navires (probabilités truquées).
- **Cartes "Risque"** : Ajout de sortilèges à double tranchant (ex : 50% de chance de toucher l'ennemi, 50% de chance de se blesser soi-même).

**Architecture Technique**

- **Multijoueur Réseau** : Utiliser les sockets ('socket') pour permettre à deux humains de s'affronter sur deux machines différentes.
- **Sauvegarde (Sérialisation)** : Implémenter un système de sauvegarde (via JSON ou Pickle) pour permettre au joueur de reprendre une partie interrompue.

**Conclusion finale :** *Wizard Battleship* a été une expérience formatrice qui nous a permis de consolider nos acquis en Python tout en découvrant les contraintes d'un projet réel. Nous sommes fiers d'avoir livré un jeu fonctionnel, stable et doté d'une véritable identité visuelle.

## 9 Annexes

### 9.1 Guide d'Installation et d'Exécution

(Contenu extrait du fichier *README.md* du projet)

#### 1. Prérequis Techniques

- **Python 3.11+** doit être installé sur votre machine.
- Une **connexion internet active** est requise pour permettre à l'IA de réfléchir et de discuter (API Hugging Face).

**2. Récupération du Projet** Ouvrez un terminal et clonez le dépôt (ou extrayez l'archive fournie) :

```
git clone https://github.com/Abdoulaye-collab/PROJET-L2.git  
cd PROJET-L2
```

**3. Installation des Dépendances** Le jeu nécessite `pygame` pour le moteur graphique et `huggingface_hub` pour l'intelligence artificielle. Installez-les via pip :

```
pip install pygame huggingface_hub
```

**4. Configuration de l'IA** *Note pour la correction* : Une clé API Hugging Face valide est déjà intégrée dans le code source (`ai_llm.py`). Vous n'avez aucune configuration à effectuer : le module de chat et la stratégie avancée de l'IA fonctionneront immédiatement.

**5. Lancer le Jeu** Une fois dans le dossier du projet, lancez simplement la commande :

```
python main.py
```

(*Note : Sur certains systèmes Mac/Linux, utilisez `python3 main.py`*)

**6. Commandes et Contrôles** Le jeu se joue intégralement à la souris pour une fluidité maximale.

Action	Commande	Contexte
<b>Placer un navire</b>	Clic Gauche	Phase de Placement
<b>Pivoter un navire</b>	Touche R	Phase de Placement
<b>Tirer</b>	Clic Gauche	Phase de Combat (Grille Ennemie)
<b>Activer une Carte</b>	Clic Gauche sur la carte	Phase de Combat (Inventaire)
<b>Utiliser la Carte</b>	Clic Gauche	Phase de Combat (Grille Ennemie)
<b>Annuler la Carte</b>	Clic Gauche sur la carte	Phase de Combat (Désélection)
<b>Quitter</b>	Croix de la fenêtre	À tout moment

## Dépannage rapide

- **Erreur ModuleNotFoundError** : Vérifiez que vous avez bien lancé la commande `pip install` de l'étape 3.
- **Pas de son** : Vérifiez que vos haut-parleurs sont activés (le jeu utilise Pygame Mixer).
- **L'IA ne répond pas** : Vérifiez votre connexion internet. Si le réseau de l'université bloque les API externes, essayez en partage de connexion via un téléphone.

## 9.2 Configuration de l'IA (Prompt Engineering)

Pour donner une personnalité au "Léviathan", nous utilisons un dictionnaire de configuration envoyé au LLM. Voici l'extrait de code définissant la personnalité "**Méchante**" :

```
1 "Mechante": {
2     "style": "Tu es une IA arrogante, mechante et moqueuse.",
3     # Phrases pre-definies en cas de succes (HIT)
4     "hit": [
5         "HAHA ! Prends ca !",
6         "Touch ! Tu n'es pas de taille !",
7         "Encore un coup magistral de moi !"
8     ],
9     # Phrases pre-definies en cas d' chec (MISS)
10    "miss": [
11        "Tch... chanceux !",
12        "Hmph, a ne se reproduira pas.",
13        "Rate, mais profite... a ne durera pas."
14    ]
15 }
```

Extrait de ai\_personalities.py

## 9.3 Constantes d'Équilibrage (settings.py)

Le fichier `settings.py` centralise toute la configuration. Voici les valeurs réelles utilisées pour l'affichage et le gameplay :

```
1 # --- 1. DIMENSIONS DE L' CRAN ---
2 SCREEN_WIDTH = 1400
3 SCREEN_HEIGHT = 800
4 # --- 2. GAMEPLAY & GRILLES ---
5 GRID_SIZE = 10      # Grille de 10x10 cases
6 PROJECTILE_SPEED = 80 # Vitesse des animations
7 # Gestion asym trique des tailles de grille
8 CELL_SIZE_PLAYER = 35 # Grille Joueur (plus petite)
9 CELL_SIZE_IA = 55    # Grille IA (plus grande, cible principale)
10 # --- 3. PALETTE DE COULEURS (RGB) ---
11 COLOR_UI_BACKGROUND = (50, 60, 90)    # Fond bleu nuit
12 COLOR_MAGIC_PLAYER = (0, 180, 210)    # Cyan (Th me Joueur)
13 COLOR_MAGIC_ENEMY = (160, 32, 240)    # Violet (Th me IA)
14 COLOR_HIT = (255, 0, 0)                # Rouge (Impact)
15 # --- 4. TYPOGRAPHIE ---
16 FONT_NAME = 'assets/fonts/Sekuya-Regular.ttf'
17 FONT_NAME_GRIMOIRE = 'assets/fonts/MagicSchoolOne.ttf'
```

Extrait de settings.py

## 9.4 Crédits et Ressources

Ce projet a été réalisé en utilisant des outils open-source et des ressources libres de droits.

### Technologies et Développement

- **Moteur de Jeu :** Pygame (basé sur la bibliothèque SDL).
- **Intelligence Artificielle :** API d’inférence *Hugging Face* (Modèles GPT-2 ).
- **Assistance au code :** Utilisation de ChatGPT et Gemini pour le refactoring et le débogage.

### Ressources Graphiques et Sonores

- **Assets Visuels :** La majorité des illustrations (fonds) ont été générées par IA via **Google Gemini**.
- **Audio :** Effets sonores (tirs, impacts, ambiance) provenant de la banque de sons libres *Freesound.org*.
- **Typographie :** Polices d’écriture utilisées pour l’interface, provenant de *fonts.google.com* et autres typographies libres.

## 9.5 Galerie : Parcours d'une Session de Jeu

Cette galerie illustre le déroulement complet d'une partie, mettant en évidence la cohérence graphique et l'ergonomie.



FIGURE 6 – 1. Menu Principal : L'écran d'accueil plonge le joueur dans l'ambiance.

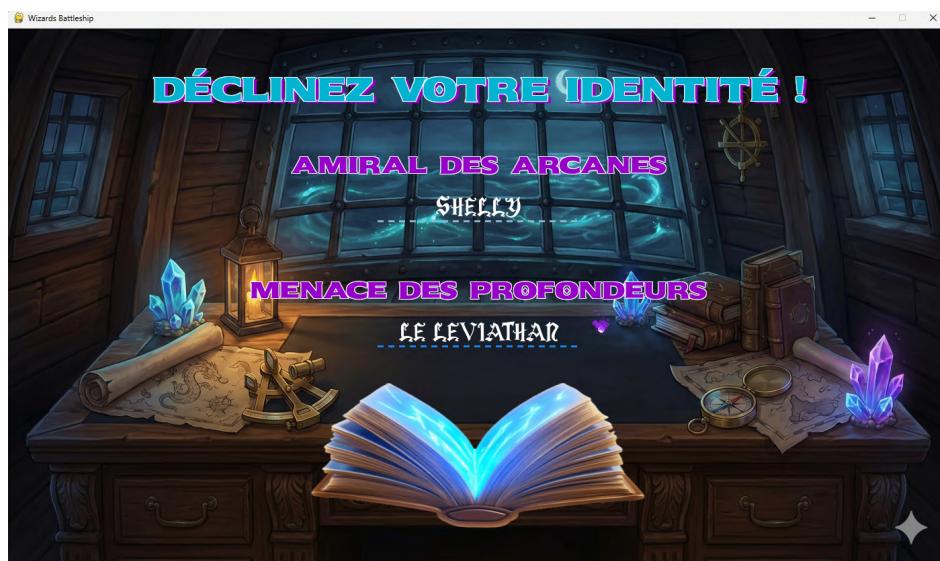


FIGURE 7 – 2. Saisie des Identités : L'orbe central s'anime pour indiquer le tour.

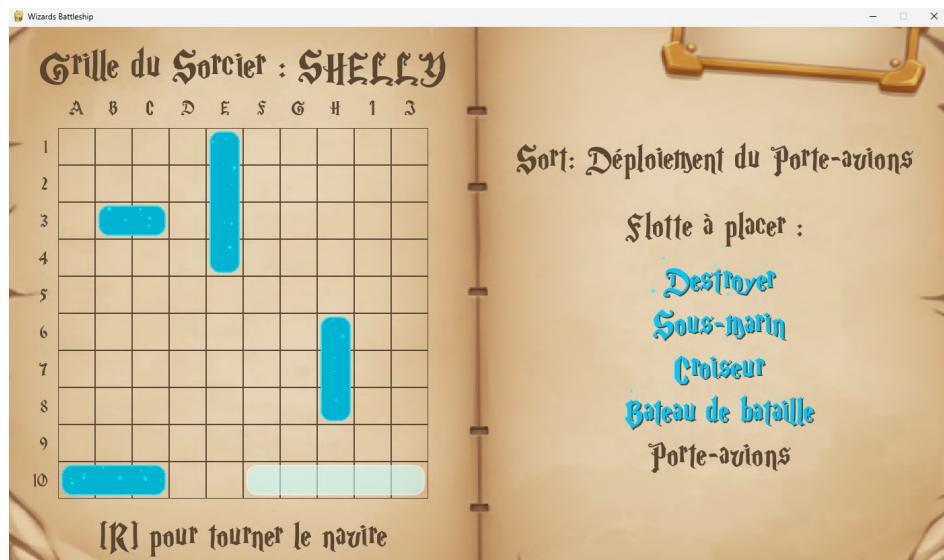


FIGURE 8 – 3. Phase de Placement : Le Drag & Drop avec prévisualisation (Vert/-Rouge).



FIGURE 9 – 4. Le Combat : Interface de tir et journal d'action en temps réel.

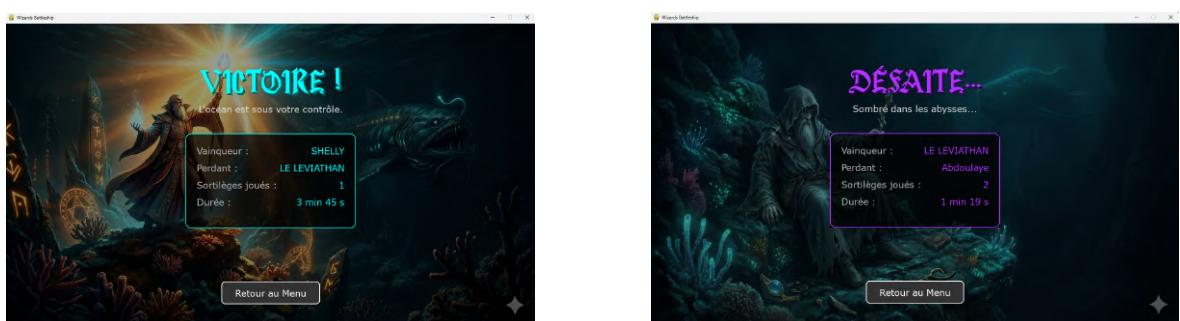


FIGURE 10 – 5. Victoire (Cyan) vs Défaite (Violet) : Adaptation dynamique du thème.