Une implémentation élégante du jeu mathématique de Grundy sur le thème des atomes.

Fonctionnalités

- 🎮 Gameplay interactif avec division des piles par glisser-déposer
- ⚛️ Visualisation atomique avec des orbites électroniques dynamiques
- 🎨 Arrière-plans en dégradé et effets de particules *magnifiques*
- 🤖 Adversaire IA pour une expérience solo
- 📊 Suivi de l'historique des mouvements
- 🔄 Gestion de l'état du jeu
- 🖥️ Redimensionnement de la fenêtre réactif

Comment jouer

1. **Début de la partie** : Le jeu commence avec un seul tas d'atomes.

2. **Effectuer des coups** :

    - Cliquez et faites glisser un atome pour le diviser
    - La distance de glissement détermine combien d'atomes se séparent
    - La division doit créer deux tas inégaux
    - *Exemple : Un tas de 7 peut être divisé en 6+1 ; 5+2 ; 4+3.*

3. **Alternance des tours** :

    - Les joueurs alternent les tours avec l'ordinateur
    - Chaque tour consiste à diviser un tas en deux parties inégales

4. **Gagner** :

    - Le jeu se termine lorsqu'aucune division valide n'est possible
    - Si vous ne pouvez pas faire de mouvement valide à votre tour, vous perdez

---

Interface en ligne de commande

```
usage: python -m grundy [-h] [--width WIDTH] [--height HEIGHT] [--piles PILES
[PILES ...]] [--scene {menu,play,gameover}]

Grundy's Game Settings

options:
  -h, --help            show this help message and exit
  --width WIDTH         set the initial screen width (default: 800)
  --height HEIGHT       set the initial screen height (default: 600)
  --piles, -p PILES [PILES ...]
                        set predefined initial piles sizes (e.g., --piles 7 5 3)
  --scene {menu,play,gameover}
                        choose the initial scene to start (default: 'menu')
```

Architecture

L'architecture du jeu repose sur une structure modulaire, permettant de séparer efficacement les différentes tâches tout en assurant une communication fluide entre les divers composants du système. Au cœur de cette architecture se trouve le système moteur `Engine`, il s'agit du coordinateur central gérant la fenêtre d'affichage, le canvas, les événements, les scènes et la logique du jeu.

Pourquoi une Approche Orientée Objet ?

1. **Encapsulation**
    - Chaque classe gère ses propres responsabilités de manière isolée
    - Réduction des dépendances et meilleure organisation du code
2. **Extensibilité**
    - Facilité d'ajout de nouvelles fonctionnalités
    - Possibilité d'hériter des classes pour des besoins spécifiques
    - Code plus structuré et plus facile à comprendre

Fenêtre et Canvas

Une approche orientée objet pour remplacer les fonctionnalités de base de Turtle a été utilisé:

Viewport

- **Équivalent Turtle**: `turtle.getcanvas().winfo_toplevel()`
- **Responsabilité**: Gestion de la fenêtre principale et des événements système

Canvas

- **Équivalent Turtle**: `turtle.getcanvas()`
- **Responsabilité**: Surface de dessin avancée

Logique du jeu

Les mécaniques fondamentales du jeu sont gérées par le système **Logic**, qui suit l'état des piles et valide les actions des joueurs selon les règles du jeu de Grundy. Lorsqu'un joueur tente de diviser une pile, le système vérifie la validité du coup : les piles résultantes doivent être de tailles inégales et la division doit respecter les règles du jeu.

Le système **Logic** est également responsable de l'alternance des tours et de l'exécution des coups. Il gère l'alternance entre le joueur humain et l'adversaire contrôlé par l'IA, valide les coups et diffuse les événements relatifs aux actions des joueurs. L'IA utilise une logique simple, choisissant au hasard des divisions valides. Le système suit l'évolution de la partie et déclare un vainqueur lorsque les conditions sont remplies.

Système d'évènements

Le système d'évènements facilite la communication entre les différents composants, sans qu'ils soient directement liés les uns aux autres. Il repose sur un modèle de publication-abonnement, où les composants

s'échangent des événements au lieu de s'appeler mutuellement. Cela réduit les dépendances entre les éléments, rendant le système plus modulaire et plus facile à maintenir.

Les événements couvrent une large gamme d'actions, allant des entrées utilisateur aux changements d'état du jeu, en passant par la gestion de la fenêtre et les conditions de victoire.

Voici une liste non-exhaustive des évènements existants:

- `WINDOW_RESIZE`: Changement de résolution
- `UPDATE`: Mis à jour du rendu
- `SCENE_CHANGED`: Transition à une nouvelle scène
- `MOVE_MADE`: Coup effectué par un joueur
- `GAME_OVER`: Fin de la partie
- `GAME_RESET`: Réinitialisation de la partie
- `PILE_ADDED/REMOVED`: Etat d'une pile mise à jour

Système de composants

Le jeu utilise une architecture basée sur des nœuds où chaque composant visuel hérite de la classe de base `Node`.

Chaque **Node** est un composant autonome ayant des fonctions spécifiques, qu'il s'agisse de la gestion d'éléments visuels ou de la logique du jeu. Les nodes gèrent leur propre cycle de vie, traitent les événements qui les concernent et gèrent leurs ressources. Cette indépendance permet une grande flexibilité, offrant la possibilité d'ajouter, de modifier ou de supprimer des composants sans perturber l'intégrité du système global.

Système de scènes

Le système **Scene** est responsable de l'organisation des états du jeu. Chaque scène représente un état spécifique, comme le menu principal, le jeu lui-même ou la fin de la partie. Les scènes servent de conteneurs pour les objets du jeu, en gérant leur cycle de vie (création et destruction).

Cela permet d'assurer des transitions fluides entre les différentes phases du jeu, tout en maintenant une organisation claire et bien définie des éléments.

Scène du Menu

La scène du menu (`scenes/menu.py`) crée une interface sur le thème industriel avec :

- Un arrière-plan en dégradé représentant le ciel
- Des visualisations de tours de refroidissement et d'une centrale électrique
- Un texte clignotant « Cliquez pour jouer »
- Une gestion du clic pour passer à la scène de jeu

Scène de Jeu

La scène principale du jeu (`scenes/play.py`) combine :

- Un arrière-plan sombre en dégradé pour créer de la profondeur

- Des particules flottantes pour l'atmosphère
- Le système de visualisation atomique
- Un suivi de l'historique des déplacements

Scène de Fin de Partie

La scène de fin de jeu (`scenes/gameover.py`) comprend :

- Un dégradé dramatique du rouge au noir
- Un message de victoire ou de défaite
- L'annonce du gagnant
- Une option pour recommencer la partie

Composants

Visualisation Atomique (`nodes/atoms`)

Les atomes sont placés aléatoirement dans une zone prédéfinie à l'aide d'un algorithme de force brute, qui semble être l'approche privilégiée dans ce contexte. L'algorithme tente `x` fois de positionner un atome sans chevauchement avec les autres. Si aucun emplacement valide n'est trouvé après ces tentatives, un avertissement graphique est affiché pour informer l'utilisateur.

Le composant de visualisation atomique est un système sophistiqué qui gère :

Structure des Atomes

- Représentation des atomes avec un noyau et des orbites électroniques
- Calcul de la distribution des électrons basé sur les principes de la mécanique quantique
- Prise en charge des orbites électroniques dynamiques avec un espacement correct

Interaction

- Permet le glisser-déposer pour diviser les atomes
- Affiche un aperçu de la division avec le nombre d'unités
- Gère la détection des collisions entre atomes
- Assure le placement des atomes dans la zone d'affichage

Animation

- Anime les orbites électroniques avec un mouvement basé sur la physique
- Ajuste la vitesse des électrons en fonction du rayon de l'orbite
- Maintient des transitions fluides et des mises à jour dynamiques

Particules (`nodes/particles.py`)

Le système de particules génère des effets atmosphériques :

- Génération de particules basée sur la densité
- Mouvement vertical des particules

- Variations d'intensité utilisant des ondes sinusoïdales

Centrale Électrique (`nodes/power_plant.py`)

- Visualisation détaillée d'une centrale électrique
- Composants de bâtiment multiples
- Détails du système de ventilation
- Échelle et positionnement appropriés

Tours de Refroidissement (`nodes/cooling_tower.py`)

- Formes réalistes des tours de refroidissement
- Proportions et placement corrects
- Intégration fluide avec l'ensemble de la scène

Historique des Mouvements (`nodes/move_history.py`)

- Affiche les derniers coups joués avec une présentation claire et lisible
- Permet de configurer la longueur de l'historique
- Écoute les événements émis par le système logique pour mettre à jour les informations

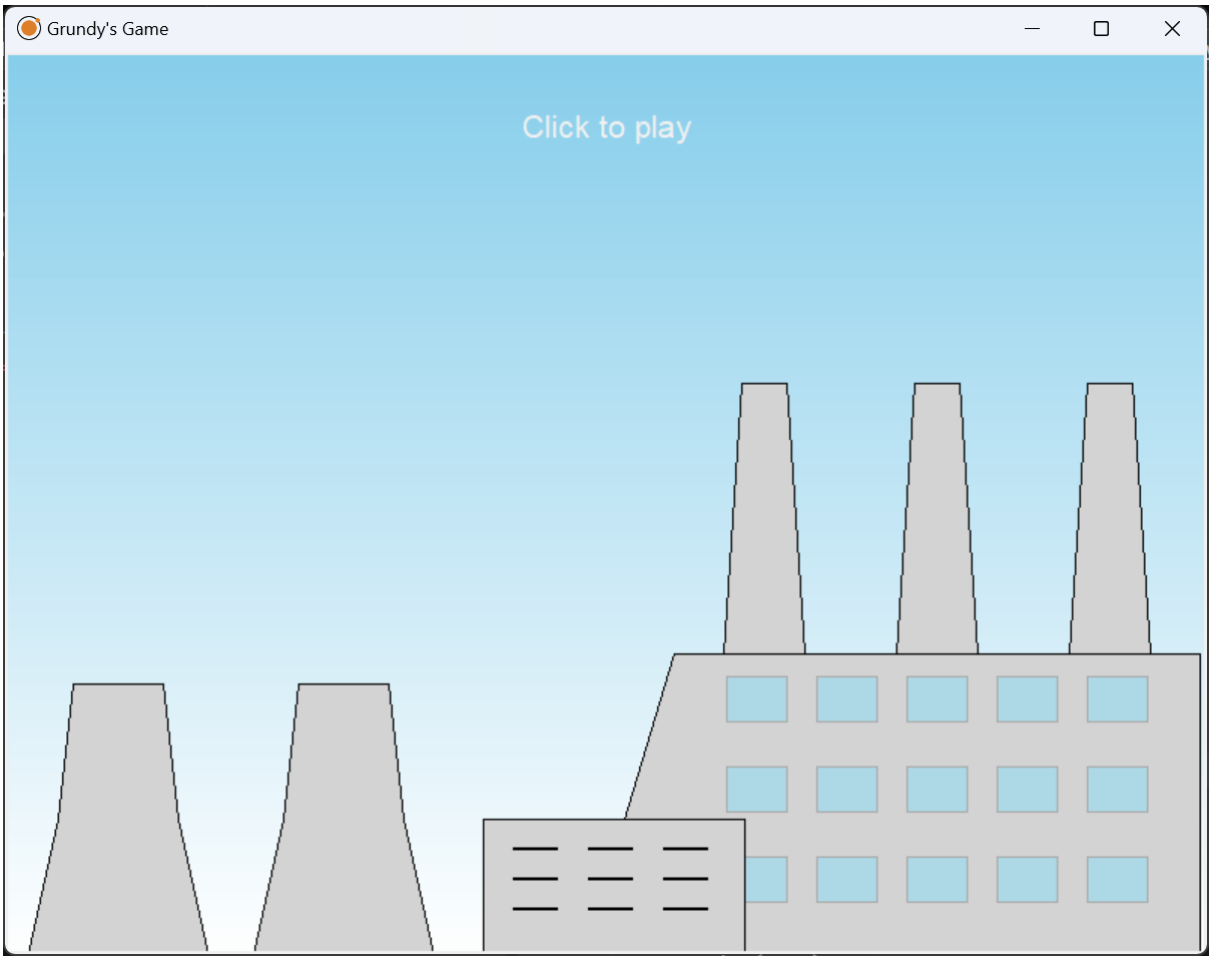
Autres Composants

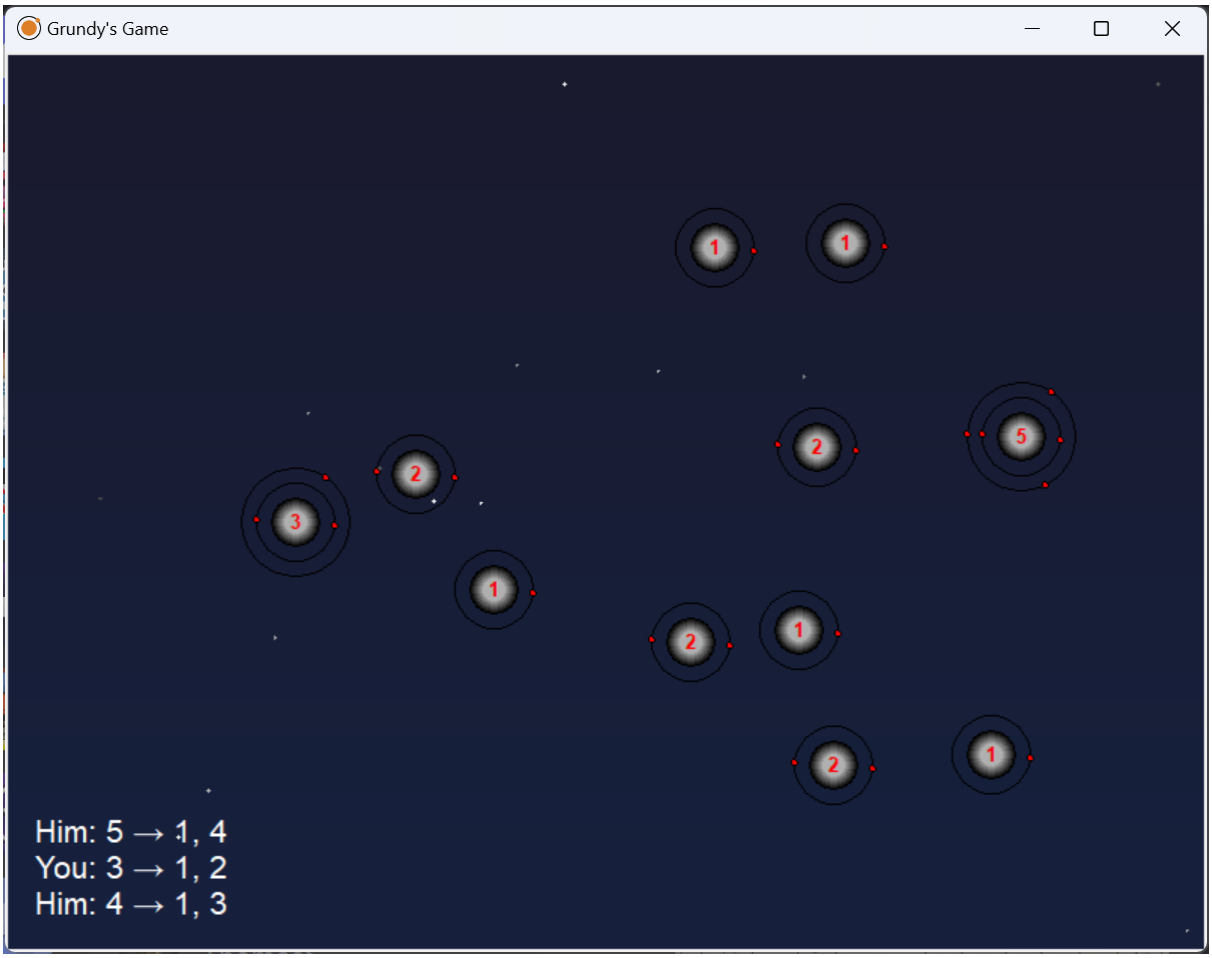
De nombreux autres composants enrichissent l'expérience de jeu, chacun ayant un rôle spécifique. Parmi eux :

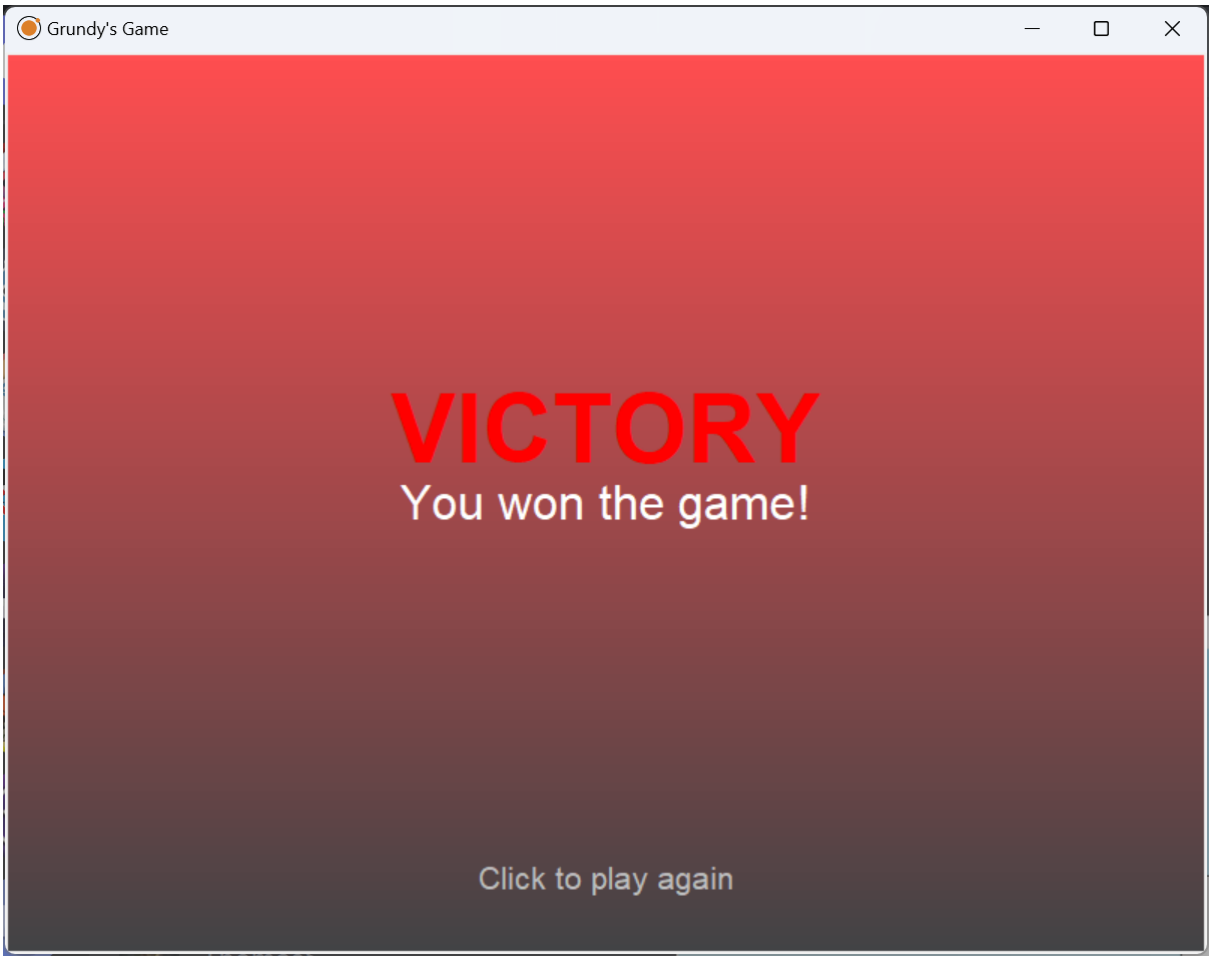
- **`nodes/flashing_text.py`** : Permet la création d'un texte personnalisé dont la couleur oscille progressivement entre une teinte de départ et une teinte d'arrivée pour attirer l'attention.
- **`nodes/gradient_background.py`** : Gère l'affichage des arrière-plans dégradés utilisés dans différentes scènes pour une immersion visuelle renforcée.
- ...

Utilitaires

- Le module `utils/colors.py` fournit des outils pour manipuler les couleurs dans le jeu.
- Le module `utils/geom.py` gère des calculs et structures géométriques utiles dans le jeu.


Grundy's Game
Click to play


Grundy's Game
1
1
2
5
2
3
1
2
1
2
1
Him: 5 → 1, 4
You: 3 → 1, 2
Him: 4 → 1, 3


Grundy's Game
VICTORY
You won the game!
Click to play again