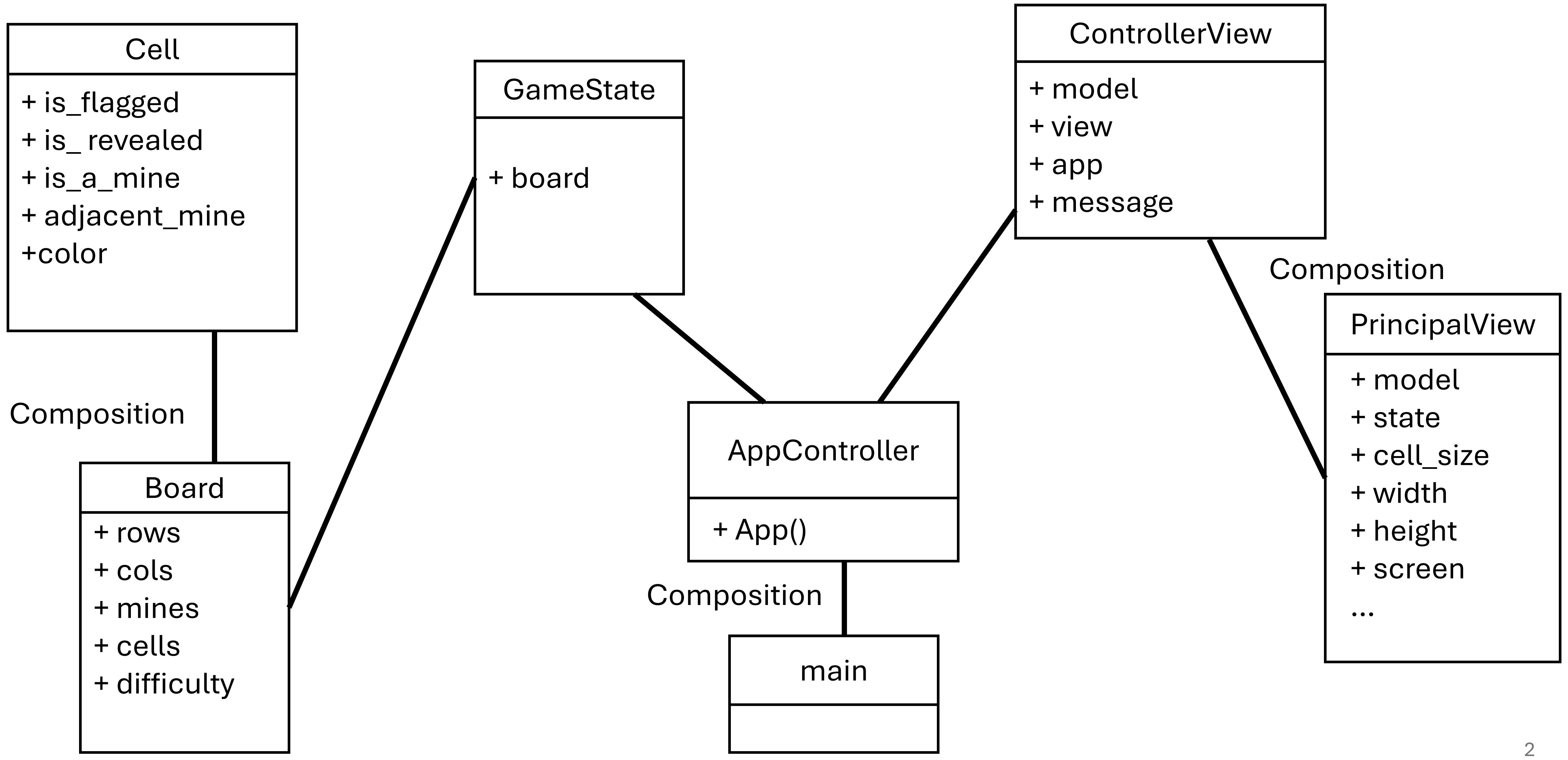
Minesweeper

Projet de Vincent et Ludivine

UML



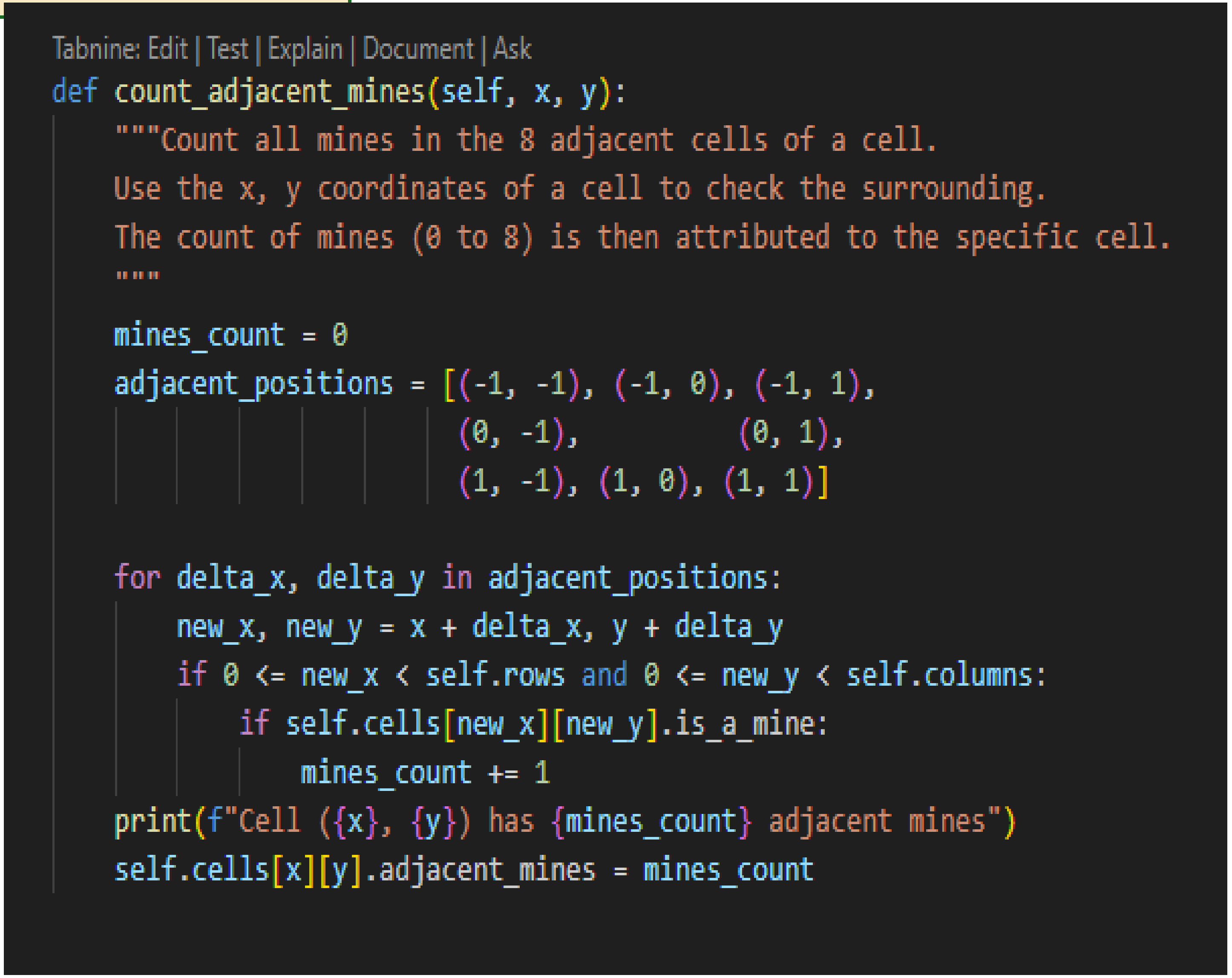
Cell
+ is_flagged
+ is_ revealed
+ is_a_mine
+ adjacent_mine
+color
Composition
Board
+ rows
+ cols
+ mines
+ cells
+ difficulty
GameState
+ board
AppController
+ App()
Composition
main
ControllerView
+ model
+ view
+ app
+ message
Composition
PrincipalView
+ model
+ state
+ cell_size
+ width
+ height
+ screen
...

Model Board.count_adjacent_mines

- Permet de voir si il y a des bombes autours d'une cellule

- Utilise delta

- Bloque la recherche sur les bords de la grille grâce à adjacent_position et à la condition zéro

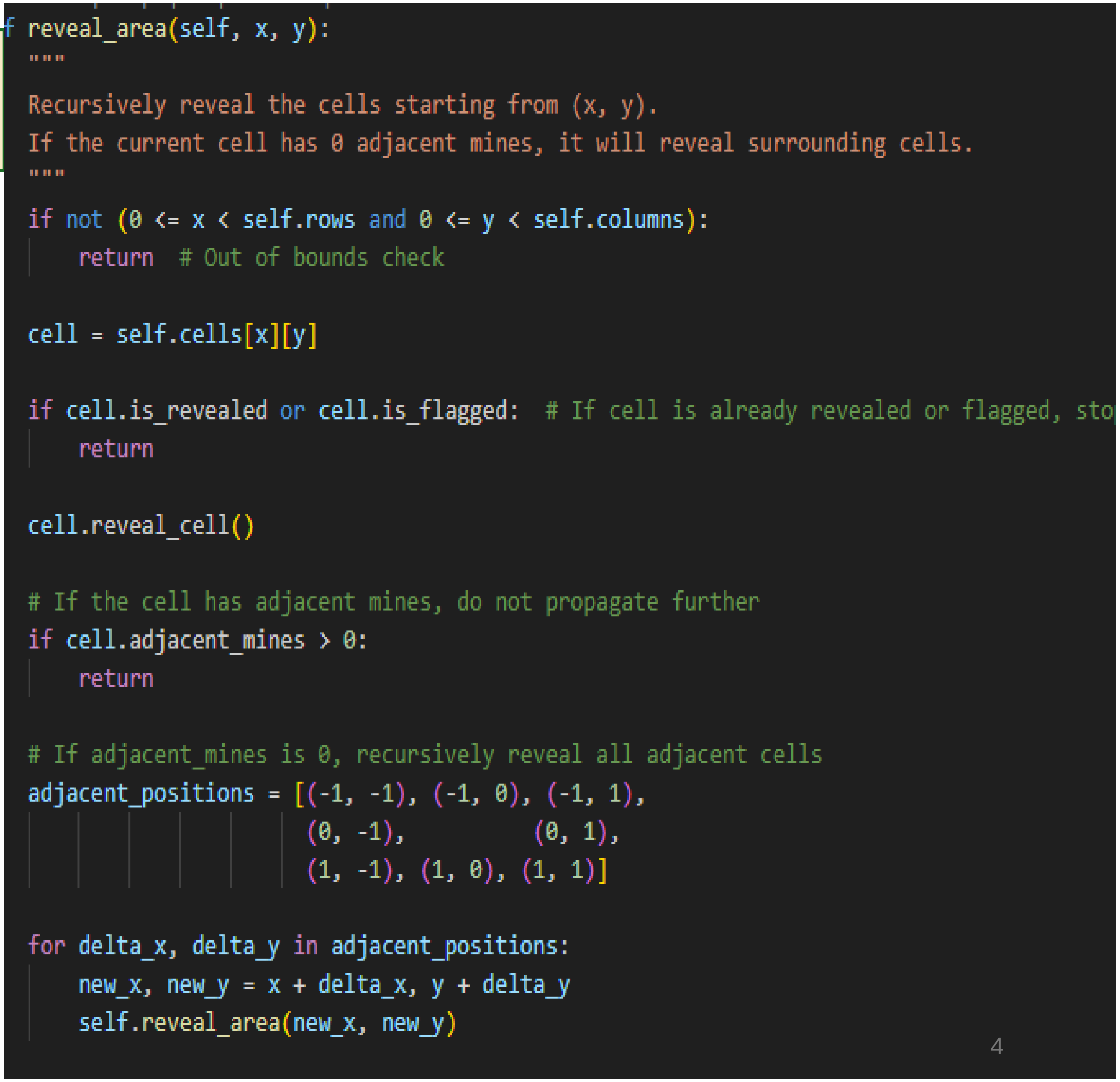


Tabnine: Edit | Test | Explain | Document | Ask

```python
def count_adjacent_mines(self, x, y):
    """Count all mines in the 8 adjacent cells of a cell.
    Use the x, y coordinates of a cell to check the surrounding.
    The count of mines (0 to 8) is then attributed to the specific cell.
    """
    mines_count = 0
    adjacent_positions = [(-1, -1), (-1, 0), (-1, 1),
                          (0, -1),           (0, 1),
                          (1, -1), (1, 0), (1, 1)]

    for delta_x, delta_y in adjacent_positions:
        new_x, new_y = x + delta_x, y + delta_y
        if 0 <= new_x < self.rows and 0 <= new_y < self.columns:
            if self.cells[new_x][new_y].is_a_mine:
                mines_count += 1
    print(f"Cell ({x}, {y}) has {mines_count} adjacent mines")
    self.cells[x][y].adjacent_mines = mines_count
```

Model Board.reveal_area

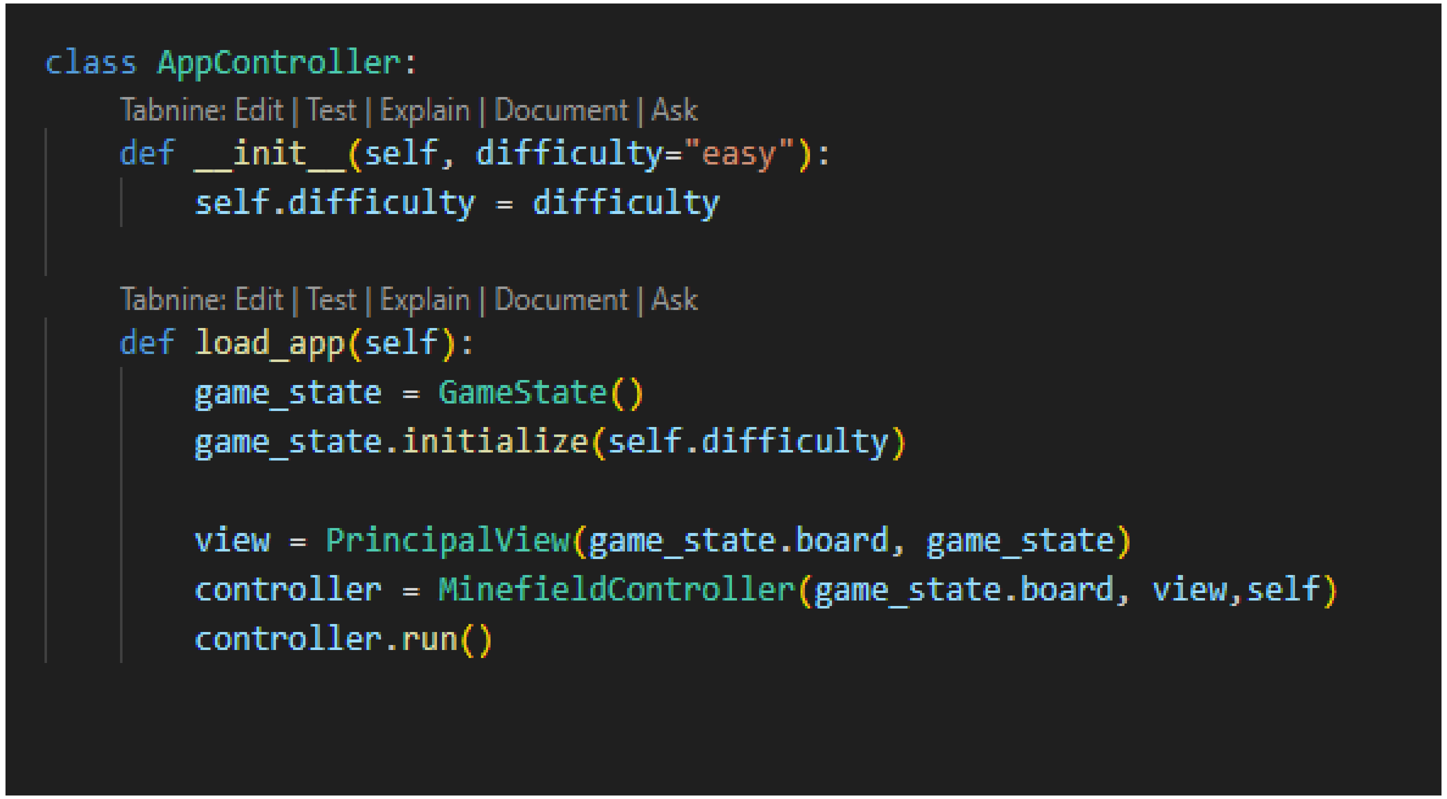
- Fonction de propagation
- Reprend la logique de la fonction pour les mines adjacentes
- Utilise la fonction reveal_cell du model Cell



```
f reveal_area(self, x, y):
    """
    Recursively reveal the cells starting from (x, y).
    If the current cell has 0 adjacent mines, it will reveal surrounding cells.
    """
    if not (0 <= x < self.rows and 0 <= y < self.columns):
        return  # Out of bounds check

    cell = self.cells[x][y]

    if cell.is_revealed or cell.is_flagged:  # If cell is already revealed or flagged, sto
        return

    cell.reveal_cell()

    # If the cell has adjacent mines, do not propagate further
    if cell.adjacent_mines > 0:
        return

    # If adjacent_mines is 0, recursively reveal all adjacent cells
    adjacent_positions = [(-1, -1), (-1, 0), (-1, 1),
                           (0, -1),          (0, 1),
                           (1, -1), (1, 0), (1, 1)]

    for delta_x, delta_y in adjacent_positions:
        new_x, new_y = x + delta_x, y + delta_y
        self.reveal_area(new_x, new_y)
```

AppController

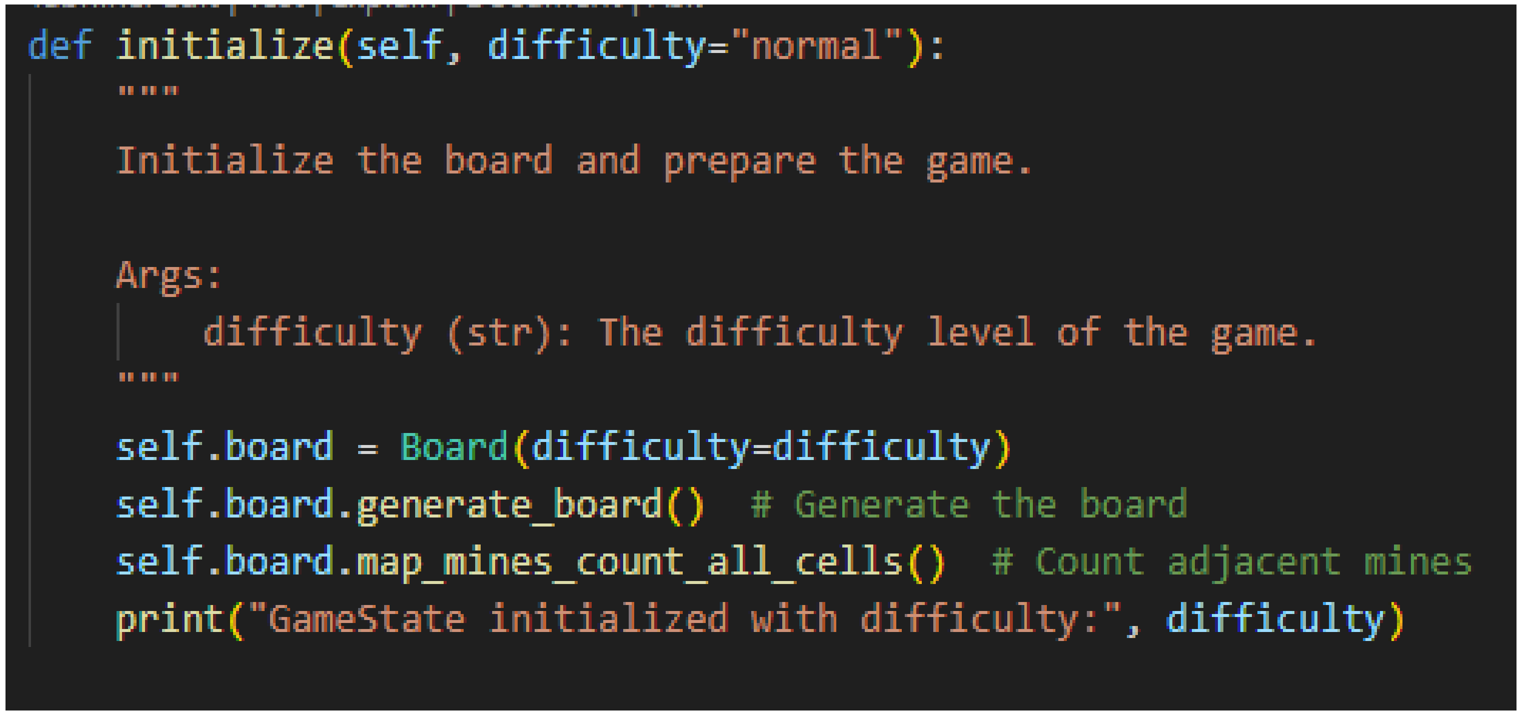
- C’est la methode qui va lancer le jeu
- Elle concentre tous les parameters pour crée le jeu
- Normalement placer dans main.py
- Choix car je veux pouvoir rappeler cette methode dans le future pour la selection de niveau



```
class AppController:
    Tabnine: Edit | Test | Explain | Document | Ask
    def __init__(self, difficulty="easy"):
        self.difficulty = difficulty

    Tabnine: Edit | Test | Explain | Document | Ask
    def load_app(self):
        game_state = GameState()
        game_state.initialize(self.difficulty)

        view = PrincipalView(game_state.board, game_state)
        controller = MinefieldController(game_state.board, view,self)
        controller.run()
```

GameState.Initialize

- Méthode pour initialiser la grille avec les mines
- Elle appelle le constructeur
- La méthode qui va créer le gille avec le Cell
- suivant la difficulté choisie elle va ajuster les paramètres de la grille



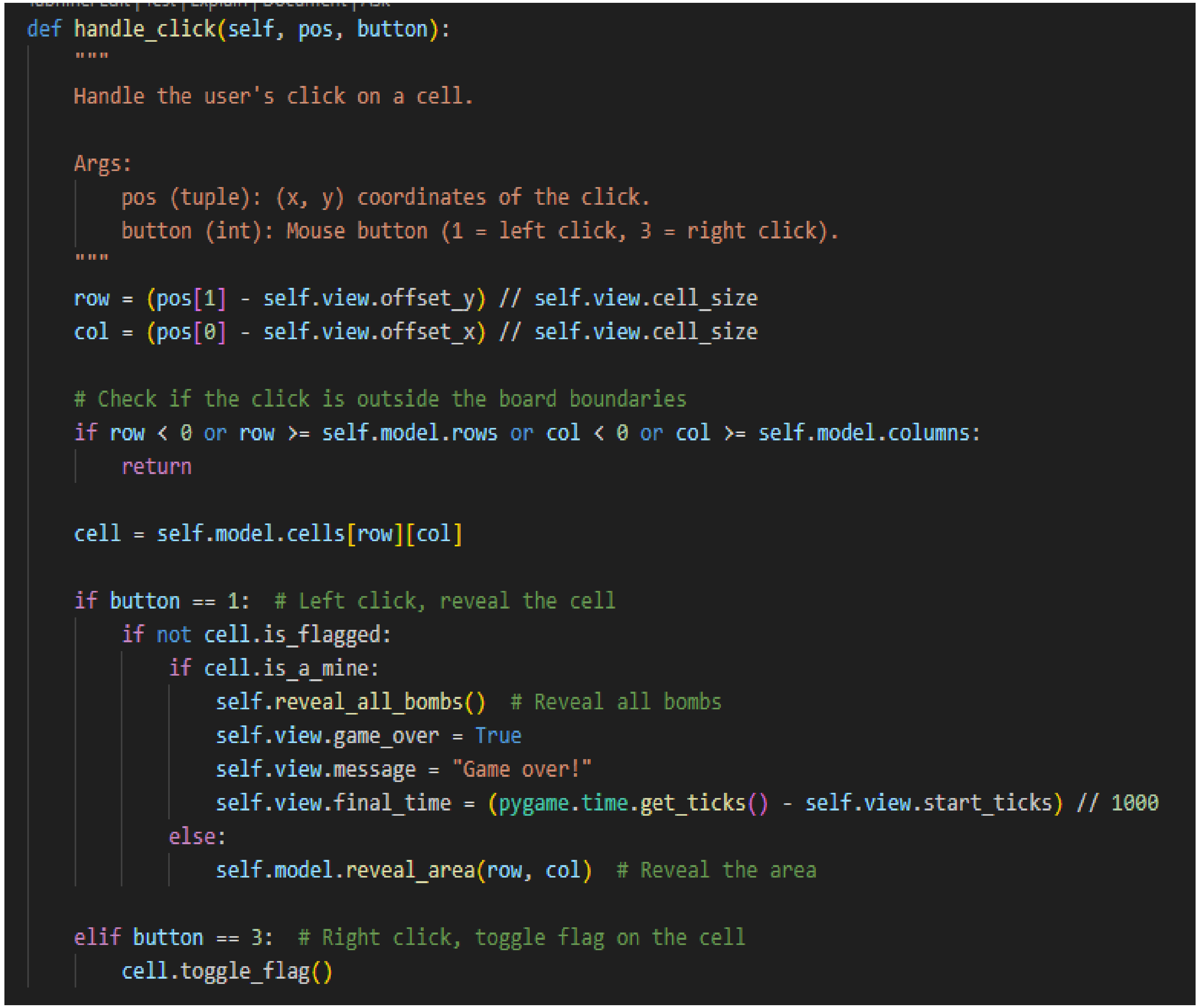
```python
def initialize(self, difficulty="normal"):
    """
    Initialize the board and prepare the game.

    Args:
        difficulty (str): The difficulty level of the game.
    """
    self.board = Board(difficulty=difficulty)
    self.board.generate_board()  # Generate the board
    self.board.map_mines_count_all_cells()  # Count adjacent mines
    print("GameState initialized with difficulty:", difficulty)
```

Controller_view. handler_click

- Il gère touts les clicks sur la vue

- Il appel les méthodes adéquates grâce aux conditions

- Il gère les clicks en dehors de la grille

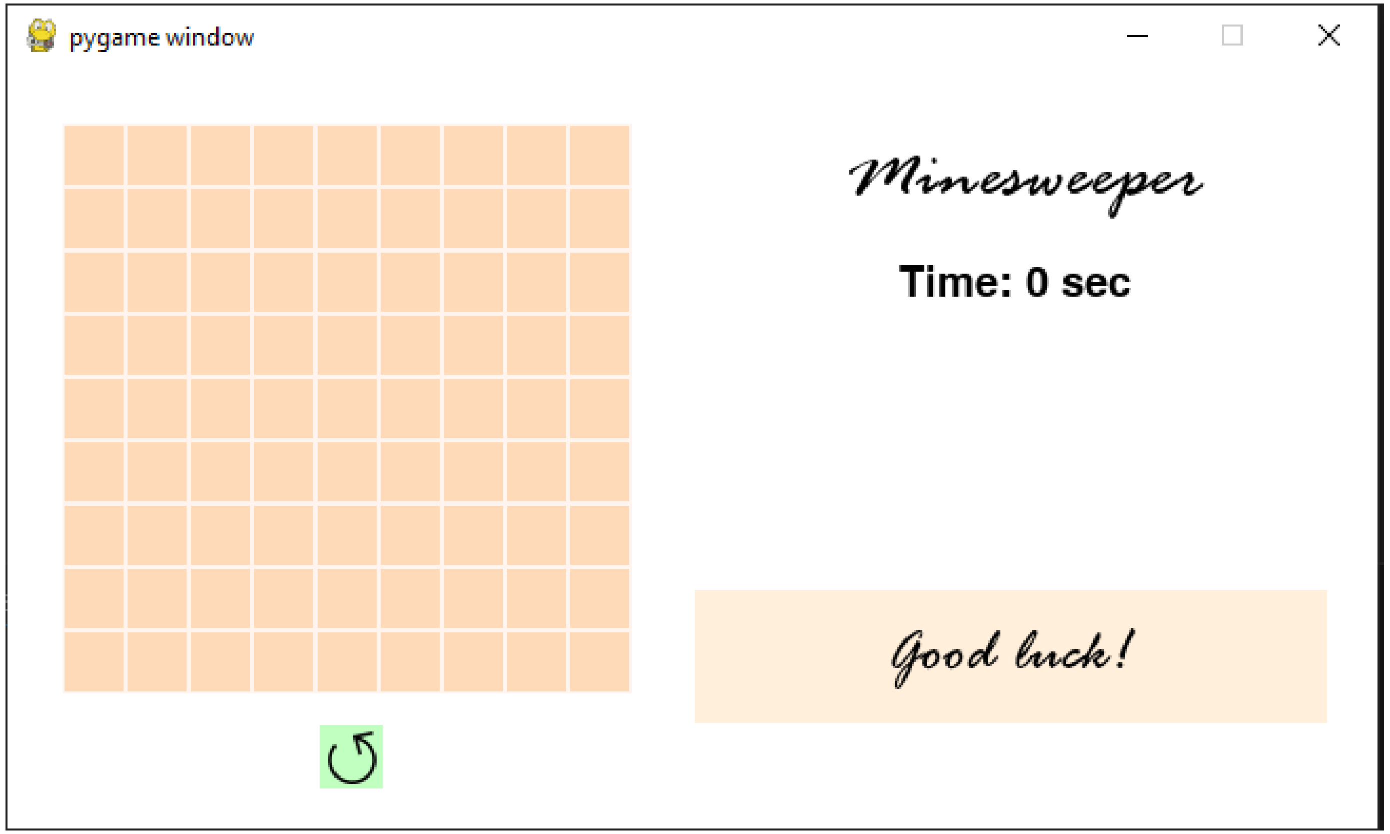


```python
def handle_click(self, pos, button):
    """
    Handle the user's click on a cell.

    Args:
        pos (tuple): (x, y) coordinates of the click.
        button (int): Mouse button (1 = left click, 3 = right click).
    """
    row = (pos[1] - self.view.offset_y) // self.view.cell_size
    col = (pos[0] - self.view.offset_x) // self.view.cell_size

    # Check if the click is outside the board boundaries
    if row < 0 or row >= self.model.rows or col < 0 or col >= self.model.columns:
        return

    cell = self.model.cells[row][col]

    if button == 1:  # Left click, reveal the cell
        if not cell.is_flagged:
            if cell.is_a_mine:
                self.reveal_all_bombs()  # Reveal all bombs
                self.view.game_over = True
                self.view.message = "Game over!"
                self.view.final_time = (pygame.time.get_ticks() - self.view.start_ticks) // 1000
            else:
                self.model.reveal_area(row, col)  # Reveal the area

    elif button == 3:  # Right click, toggle flag on the cell
        cell.toggle_flag()
```

Difficultés rencontrés

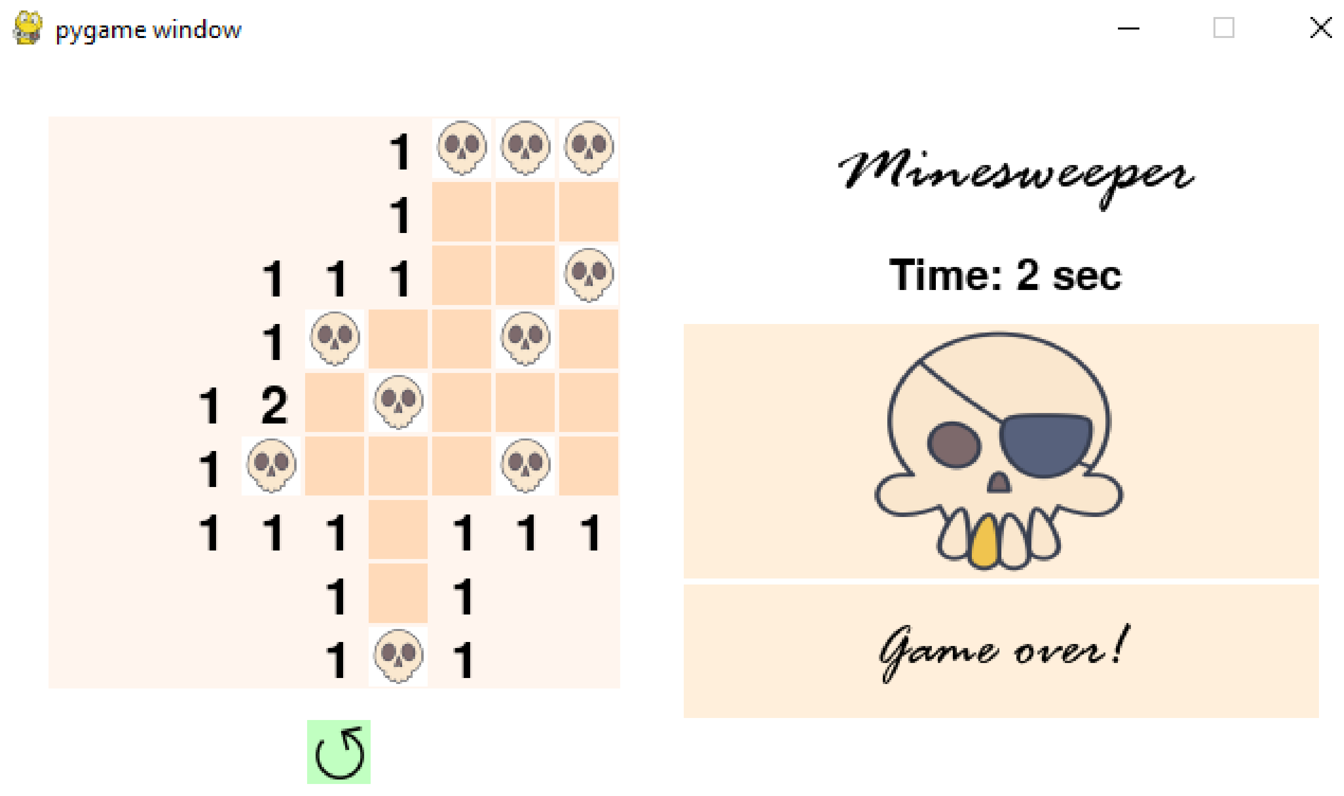
- Relier le back et pygame
  - Parfois pas intuitif

- La fonction de propagation et la récursivité



Améliorations

- Rajouter la sélection du niveau
- Rajouter le compteur de bombes restantes et présentes
- Bloquer le flag quand il n'y a plus de bombes
- Cosmétiques : animations et sons


pygame window
Minesweeper
Time: 2 sec
Game over!