Розробка гри Тетріс на Python

Використання бібліотек pygame та numpy

Документація коду

26 травня 2025 р.

Зміст

1 Вступ

Тетріс — це класична головоломка, яка вимагає стратегічного мислення та швидкої реакції. У цьому документі розглядається реалізація гри Тетріс на мові програмування Python з використанням бібліотек pygame та numpy.

2 Імпорт необхідних бібліотек

На початку коду імпортуються необхідні бібліотеки:

- **pygame** — бібліотека для створення графічного інтерфейсу, керування вікнами, обробки подій і малювання графіки. Вона є основою для візуалізації гри.
- **numpy** — потужна бібліотека для числових обчислень. Тут вона використовується для роботи з формами тетріс-фігур у вигляді масивів.
- **random** — використовується для випадкового вибору наступної фігури.
- **sys та traceback** — дозволяють коректно обробляти винятки та закривати програму у разі помилки.

```
import pygame
import numpy as np
import random
import sys
import traceback
```

Лістинг 1: Імпорт бібліотек

У цьому коді `pygame.init()` використовується для ініціалізації всіх модулів бібліотеки, а також визначаються основні параметри гри: розміри клітинки, кількість стовпців і рядків, розмір вікна, кількість кадрів на секунду тощо.

3 Визначення констант

Константи, такі як `CELL`, `COLS` та `ROWS`, визначають розміри ігрової сітки та клітинок:

```
CELL = 30
COLS, ROWS = 10, 20
WIDTH, HEIGHT = CELL * COLS, CELL * ROWS
PREVIEW_WIDTH = 170
TOTAL_WIDTH = WIDTH + PREVIEW_WIDTH
FPS = 60
```

Лістинг 2: Основні константи гри

Пояснення констант:

- `CELL = 30` — розмір однієї клітинки у пікселях.
- `COLS = 10, ROWS = 20` — стандартна кількість стовпців і рядків для поля тетрісу.
- `WIDTH, HEIGHT` — ширина і висота основного ігрового поля.

- `PREVIEW_WIDTH` — ширина області для попереднього перегляду наступної фігури.
- `TOTAL_WIDTH` — загальна ширина вікна гри.
- `FPS = 60` — кількість кадрів на секунду, що забезпечує плавність анімації.

Таке винесення параметрів в окремі константи дозволяє легко змінювати конфігурацію гри без потреби вносити зміни в основну логіку. Це підвищує масштабованість і зручність супроводу коду.

4 Палітра кольорів

У коді визначено палітру кольорів у вигляді списку кортежів, де кожен кортеж — це значення кольору у форматі RGB:

```
colors = [
    (30, 30, 30),
    (0, 255, 255),
    (0, 0, 255),
    (255, 165, 0),
    (255, 255, 0),
    (0, 255, 0),
    (128, 0, 128),
    (255, 0, 0)
]
```

Лістинг 3: Палітра кольорів для тетріс-фігур

Кожен елемент у списку відповідає певному типу тетріс-фігури. Кольори задаються у форматі (Red, Green, Blue). Перший колір — це базовий фон сітки гри (темно-сірий). Інші — яскраві кольори для кожної з фігур.

Такий підхід дозволяє легко змінювати кольорову гаму гри або адаптувати її під різні стилі інтерфейсу. Це покращує візуальне сприйняття ігрового процесу.

5 Форми тетріс-фігур

Форми тетріс-фігур представлені у вигляді масивів NumPy, де кожне число вказує на наявність блоку певного типу:

```
shapes = [
    np.array([[1, 1, 1, 1]]),
    np.array([[2, 0, 0], [2, 2, 2]]),
    np.array([[0, 0, 3], [3, 3, 3]]),
    np.array([[4, 4], [4, 4]]),
    np.array([[0, 5, 5], [5, 5, 0]]),
    np.array([[0, 6, 0], [6, 6, 6]]),
    np.array([[7, 7, 0], [0, 7, 7]])
]
```

Лістинг 4: Визначення форм тетріс-фігур

Цілі числа (1–7) відповідають типам фігур, що пов'язуються з кольорами. Нуль позначає порожню клітинку в масиві.

Завдяки використанню NumPy, можлива проста обробка фігур:

- обертання (`np.rot90()`),
- перевірка допустимості розміщення (`valid()`),
- копіювання чи порівняння форм.

Така структурована модель забезпечує гнучкість і ефективність у керуванні фігурами під час гри.

6 Допоміжні функції

6.1 Визначення індексу фігури

Фігури зберігаються у вигляді двовимірних масивів NumPy, де кожна форма визначає свою конфігурацію та індекс кольору. Це дозволяє ефективно працювати з фігурами під час гри.

Щоб визначити, яка саме фігура використовується в даний момент, застосовується функція:

```
def get_shape_index(current):
    for i, shape in enumerate(shapes):
        if np.array_equal(current, shape):
            return i
    raise ValueError("Shape not found in shapes list")
```

Лістинг 5: Функція визначення індексу фігури

`get_shape_index()` повертає індекс поточної фігури в списку `shapes`. Це значення потрібне для визначення кольору (`colors[index + 1]`) і оновлення гри при появі нової фігури.

Такий підхід робить код гнучким і дозволяє легко відстежувати, з якою фігурою зараз працює гравець.

6.2 Функція обертання

Функція `rotate` відповідає за обертання фігури на 90 градусів проти годинникової стрілки за допомогою методу `np.rot90` з бібліотеки NumPy:

```
def rotate(shape):
    return np.rot90(shape, k=-1)
```

Лістинг 6: Функція обертання фігури

`np.rot90(shape, k=-1)` обертає масив `shape` на 90° проти годинникової стрілки. `k=-1` означає одне обертання в негативному напрямку (тобто проти годинникової). Повертається нова орієнтація фігури без зміни оригінального об'єкта.

Механізм обертання є критично важливим у тетрісі, оскільки дозволяє гравцям стратегічно розміщувати фігури на полі, особливо у вузьких просторах.

6.3 Перевірка валідності позиції

Функція `valid` перевіряє, чи може фігура зайняти певну позицію на ігровому полі. Вона відіграє ключову роль у русі й обертанні фігур.

```python
def valid(grid, shape, offset):
    off_x, off_y = offset
    for y in range(shape.shape[0]):
        for x in range(shape.shape[1]):
            if shape[y, x]:
                if x + off_x < 0 or x + off_x >= COLS or y + off_y >=
    ROWS:
                    return False
                if grid[y + off_y, x + off_x]:
                    return False
    return True
```

Лістинг 7: Функція перевірки валідності позиції

Основні перевірки:

- Межі поля: чи не виходить фігура за лівий, правий або нижній край.
- Колізії: чи не перетинається з уже розміщеними блоками.
- Позиція: задається як зсув `offset (x, y)`, що відповідає поточному місцю розміщення фігури.

Ця функція забезпечує правильну логіку гри — фігури не можуть «залазити» одна на одну або виходити за межі поля.

6.4 Очищення заповнених рядків

Функція `clear_rows` відповідає за пошук та видалення повністю заповнених рядків. Це ключовий механізм тетрісу, який впливає на набір очок та динаміку гри.

```python
def clear_rows(grid):
    non_full_rows = grid[np.any(grid == 0, axis=1)]
    cleared = ROWS - non_full_rows.shape[0]
    new_rows = np.zeros((cleared, COLS), dtype=int)
    return np.vstack([new_rows, non_full_rows]), cleared
```

Лістинг 8: Функція очищення заповнених рядків

Принцип роботи:

- `np.any(grid == 0, axis=1)` знаходить всі рядки, де є хоча б одна порожня клітинка (неповні рядки).
- Повні рядки видаляються, а замість них додаються нові порожні з самого верху (`new_rows`).
- `cleared` — кількість очищених рядків, що потім використовується для нарахування очок та підвищення рівня складності.

7 Функції відображення

7.1 Малювання сітки

Функція `draw_grid` створює візуальну структуру гри, малюючи сітку на ігровій поверхні:

```
def draw_grid(surface):
    for y in range(ROWS):
        for x in range(COLS):
            pygame.draw.rect(surface, (60, 60, 60),
                             (x * CELL, y * CELL, CELL, CELL), 1)
```

Лістинг 9: Функція малювання сітки

Вона ітерується по кожній клітинці сітки та малює контури прямокутників, що задає чітке і зрозуміле ігрове поле для падаючих фігур.

7.2 Малювання одного блоку

Функція `draw_block` відповідає за візуалізацію одного окремого блоку (квадрата), що входить до складу фігури або розташований у сітці:

```
def draw_block(surface, color, rect):
    pygame.draw.rect(surface, color, rect, border_radius=6)
    pygame.draw.rect(surface, (255, 255, 255), rect, 2, border_radius=6)
```

Лістинг 10: Функція малювання одного блоку

Принцип роботи:

- **Основне заливання** — на переданій поверхні (`surface`) малюється прямокутник кольору `color`, який відповідає кольору фігури. Використовується параметр `border_radius=6` для округлення кутів, що робить вигляд м'яким і сучасним.
- **Біла рамка** — поверх основного блоку накреслюється біла рамка товщиною 2 пікселі. Це створює ефект об'єму й допомагає чітко відокремити блоки один від одного.

Призначення: ця функція виконує ключову роль у графічному представленні гри — усі фігури і заповнені комірки на сітці складаються з таких блоків. Завдяки заливці та обводці блоки виглядають яскраво, контрастно й чітко — це значно покращує візуальне сприйняття під час гри.

7.3 Відображення фігури-привида

Функція `draw_ghost` показує «привид» фігури — візуальне підказування, де поточна фігура впаде:

```
def draw_ghost(screen, grid, shape, pos, color):
    ghost_pos = pos[:]
    while valid(grid, shape, [ghost_pos[0], ghost_pos[1] + 1]):
        ghost_pos[1] += 1
    for y in range(shape.shape[0]):
        for x in range(shape.shape[1]):
            if shape[y, x]:
                rect = pygame.Rect((ghost_pos[0] + x) * CELL,
                                   (ghost_pos[1] + y) * CELL, CELL, CELL)
                s = pygame.Surface((CELL, CELL), pygame.SRCALPHA)
                s.fill((*colors[color][:3], 40))
                screen.blit(s, rect.topleft)
```

Лістинг 11: Функція відображення фігури-привида

Це допомагає гравцям планувати ходи, показуючи фігуру в прозорому (приглушеному) кольорі. Такий ефект покращує ігровий досвід і прийняття рішень під час гри.

Принцип роботи:

- Функція визначає найнижчу позицію, куди може впасти фігура, без зіткнень.
- Потім малює її з прозорістю, щоб не плутати з активною фігурою.
- Це дає гравцеві чітке уявлення про можливий результат руху.

7.4 Прев'ю наступної фігури

Функція `draw_next_piece` відображає прев'ю наступної фігури, яка з'явиться у грі:

```
def draw_next_piece(screen, next_shape, next_color):
    preview_x = WIDTH + 20
    preview_y = 107

    font = pygame.font.SysFont('consolas', 20)
    text = font.render('NEXT:', True, (255, 255, 255))
    screen.blit(text, (preview_x, preview_y - 23))

    pygame.draw.rect(screen, (40, 40, 40),
                     (preview_x - 5, preview_y - 5, 110, 110))
    pygame.draw.rect(screen, (100, 100, 100),
                     (preview_x - 5, preview_y - 5, 110, 110), 2)

    offset_x = (4 - next_shape.shape[1]) * CELL // 4
    offset_y = (4 - next_shape.shape[0]) * CELL // 4

    for y in range(next_shape.shape[0]):
        for x in range(next_shape.shape[1]):
            if next_shape[y, x]:
                small_rect = pygame.Rect(
                    preview_x + x * 20 + offset_x,
                    preview_y + y * 20 + offset_y,
                    20, 20
                )
                draw_block(screen, colors[next_color], small_rect)
```

Лістинг 12: Функція відображення наступної фігури

Принцип роботи:

- Визначається позиція вікна прев'ю праворуч від основного ігрового поля.
- Малюється рамка та напис «NEXT:», щоб виділити область.
- Наступна фігура відображається у зменшеному масштабі з правильним центруванням.
- Це допомагає гравцям швидко оцінити майбутні ходи та планувати гру.

7.5 Відображення стартового екрану

Функція `draw_start_screen` виводить головне меню гри перед її початком, включаючи назву гри, інструкції та підказки для гравця:

```
def draw_start_screen(screen):
    screen.fill((20, 20, 40))
    font_huge = pygame.font.SysFont('consolas', 72, bold=True)
    font_medium = pygame.font.SysFont('consolas', 24)
    font_small = pygame.font.SysFont('consolas', 18)
    font_tiny = pygame.font.SysFont('consolas', 16)
    title_text = font_huge.render('TETRIS', True, (0, 255, 255))
    title_rect = title_text.get_rect(center=(TOTAL_WIDTH // 2, HEIGHT //
    4))
    screen.blit(title_text, title_rect)
    start_text = font_medium.render('Press ENTER to start', True, (255,
   255, 255))
    start_rect = start_text.get_rect(center=(TOTAL_WIDTH // 2, HEIGHT //
    2))
    screen.blit(start_text, start_rect)
    controls = [
        "CONTROLS:",
        "         - Move left/right",
        "     - Soft drop",
        "     - Rotate",
        "SPACE - Hard drop"
    ]
    for i, line in enumerate(controls):
        color = (255, 255, 0) if i == 0 else (200, 200, 200)
        font = font_small if i == 0 else font_tiny
        text = font.render(line, True, color)
        text_rect = text.get_rect(center=(TOTAL_WIDTH // 2, HEIGHT // 2
   + 80 + i * 25))
        screen.blit(text, text_rect)
    exit_text = font_tiny.render('ESC - Exit', True, (150, 150, 150))
    exit_rect = exit_text.get_rect(center=(TOTAL_WIDTH // 2, HEIGHT -
   50))
    screen.blit(exit_text, exit_rect)
```

Лістинг 13: Функція відображення стартового екрану

Принцип роботи:

- **Заливка фону:** Темно-синім кольором заповнюється екран для створення атмосферного вигляду.
- **Назва гри:** Виводиться великим шрифтом слово «TETRIS» в центрі верхньої частини екрану.
- **Підказка запуску:** В центрі екрану виводиться напис «Press ENTER to start».
- **Інструкції з керування:** Показуються клавіші керування фігурами, з виділенням заголовка жовтим кольором.
- **Підказка виходу:** Внизу екрану вказано клавішу ESC для виходу з гри.

7.6 Відображення екрану поразки

Функція `draw_game_over` створює затемнений екран і виводить підсумкові результати після завершення гри:

```
def draw_game_over(screen, score, level):
    overlay = pygame.Surface((TOTAL_WIDTH, HEIGHT))
    overlay.set_alpha(180)
    overlay.fill((0, 0, 0))
    screen.blit(overlay, (0, 0))
    box_width = 400
    box_height = 250
    box_x = (TOTAL_WIDTH - box_width) // 2
    box_y = (HEIGHT - box_height) // 2
    pygame.draw.rect(screen, (40, 40, 40), (box_x, box_y, box_width, box_height), border_radius=15)
    pygame.draw.rect(screen, (255, 255, 255), (box_x, box_y, box_width, box_height), 3, border_radius=15)
    font_large = pygame.font.SysFont('consolas', 48, bold=True)
    font_medium = pygame.font.SysFont('consolas', 32)
    font_small = pygame.font.SysFont('consolas', 20)
    game_over_text = font_large.render('GAME OVER', True, (255, 50, 50))
    game_over_rect = game_over_text.get_rect(center=(TOTAL_WIDTH // 2, box_y + 50))
    screen.blit(game_over_text, game_over_rect)
    score_text = font_medium.render(f'Final score: {score}', True, (255, 255, 255))
    score_rect = score_text.get_rect(center=(TOTAL_WIDTH // 2, box_y + 110))
    screen.blit(score_text, score_rect)
    level_text = font_medium.render(f'Level reached: {level}', True, (255, 255, 255))
    level_rect = level_text.get_rect(center=(TOTAL_WIDTH // 2, box_y + 150))
    screen.blit(level_text, level_rect)
    restart_text = font_small.render('Press R to restart', True, (200, 200, 200))
    restart_rect = restart_text.get_rect(center=(TOTAL_WIDTH // 2, box_y + 190))
    screen.blit(restart_text, restart_rect)
    quit_text = font_small.render('or ESC to exit', True, (200, 200, 200))
    quit_rect = quit_text.get_rect(center=(TOTAL_WIDTH // 2, box_y + 210))
    screen.blit(quit_text, quit_rect)
```

Лістинг 14: Функція відображення екрану поразки

Принцип роботи:

- **Затемнення екрану:** Створюється прозора чорна накладка, яка накриває весь екран, щоб привернути увагу до центру.
- **Центральне вікно повідомлення:** Малюється затемнене вікно з білою рамкою, в якому буде текст.
- **Заголовок «GAME OVER»:** Великими червоними літерами виводиться повідомлення про завершення гри.

- **Результати гравця:** Виводяться фіналізовані очки (score) та досягнутий рівень (level).
- **Підказки щодо подальших дій:** Гравцеві пропонується перезапустити гру клавішею R або вийти, натиснувши ESC.

Ці функції є важливою частиною інтерфейсу гри. Вони роблять гру доступною, зрозумілою і приємною для користувача з самого початку й до її завершення.

7.7 Головна функція відображення

Функція `draw` — основна функція рендерингу, яка оновлює весь екран гри в кожному кадрі:

```
def draw(screen, grid, shape, pos, color, next_shape, next_color, score, level):
    screen.fill(colors[0])

    draw_ghost(screen, grid, shape, pos, color)

    for y in range(shape.shape[0]):
        for x in range(shape.shape[1]):
            if shape[y, x]:
                draw_block(screen, colors[color], pygame.Rect((pos[0] + x) * CELL, (pos[1] + y) * CELL, CELL, CELL))

    for y in range(ROWS):
        for x in range(COLS):
            if grid[y, x]:
                draw_block(screen, colors[grid[y, x]], pygame.Rect(x * CELL, y * CELL, CELL, CELL))

    draw_grid(screen)
    pygame.draw.rect(screen, (200, 200, 200), (0, 0, WIDTH, HEIGHT), 3)

    draw_next_piece(screen, next_shape, next_color)

    font = pygame.font.SysFont('consolas', 24)
    score_text = font.render(f'Score: {score}', True, (255, 255, 255))
    level_text = font.render(f'Level: {level}', True, (255, 255, 255))
    screen.blit(score_text, (WIDTH + 20, 20))
    screen.blit(level_text, (WIDTH + 20, 50))

    font_small = pygame.font.SysFont('consolas', 16)
    controls = [
        "Controls:",
        "        Move",
        "    Soft drop",
        "    Rotate",
        "Space Hard drop"
    ]

    for i, line in enumerate(controls):
        text = font_small.render(line, True, (200, 200, 200))
        screen.blit(text, (WIDTH + 20, 250 + i * 20))

```

```
    pygame.display.flip()
```

Лістинг 15: Головна функція відображення ігрового процесу

Функція `draw` — основна функція рендерингу, яка оновлює весь екран гри в кожному кадрі. Вона малює поточну фігуру, фігури на сітці, "привида наступну фігуру, сітку, рахунок, рівень і текст інструкцій.

Принцип роботи:

- **Очищення екрану** — заповнює фон темним кольором `colors[0]`.
- **Малювання фігури-привида** — показує прозору підказку розміщення поточної фігури.
- **Малювання активної фігури** — відображає поточну фігуру на позиції `pos` відповідним кольором.
- **Малювання застиглих фігур** — проходить по сітці `grid` і малює всі заповнені блоки.
- **Малювання сітки та рамки** — додає візуальну структуру та межі ігрового поля.
- **Прев'ю наступної фігури** — відображає майбутню фігуру у спеціальній області.
- **Відображення статистики** — показує поточний рахунок та рівень гравця.
- **Інструкції з керування** — виводить підказки щодо використання клавіш.
- **Оновлення екрану** — застосовує всі зміни через `pygame.display.flip()`.

Ця функція об'єднує всі графічні елементи гри — саме завдяки їй гравець бачить повноцінну сцену: активну фігуру, поле, наступну фігуру, підказки і рахунок. Від її роботи залежить візуальна якість гри й зручність сприйняття..

8 Ініціалізація гри

Функція `init_game` відповідає за початкову ініціалізацію гри: завантаження бібліотек, налаштування вікна та підготовку основних змінних.

```
def init_game():
    grid = np.zeros((ROWS, COLS), dtype=int)
    current = random.choice(shapes)
    next_shape = random.choice(shapes)
    color = get_shape_index(current) + 1
    next_color = get_shape_index(next_shape) + 1
    pos = [COLS // 2 - current.shape[1] // 2, 0]
    score = 0
    level = 1
    lines_cleared = 0
    fall_timer = 0
    fall_delay = 600

    return grid, current, next_shape, color, next_color, pos,
```

```
15              score, level, lines_cleared, fall_timer, fall_delay
```

Лістинг 16: Функція ініціалізації гри

Основні складові:

- Ігрова сітка — двовимірний масив, що зберігає стан кожної клітинки.
- Поточна та наступна фігури — визначаються випадково зі списку форм.
- Позиція фігури на полі — початкове розташування у верхній частині сітки.
- Рахунок і рівень — початкові параметри для відстеження прогресу.

Принцип роботи:

- Ігрова сітка створюється порожньою.
- Вибираються поточна та наступна фігури.
- Визначаються їхні кольори для відображення.
- Початкові значення рахунку, рівня та таймерів встановлюються.
- Позиція поточної фігури розміщується по центру зверху.

Цей процес створює впорядковане середовище для початку гри і забезпечує коректну роботу всіх елементів.

9 Основна функція запуску гри

Функція `main()` — це головний цикл гри, який керує її станами, обробкою подій та оновленням grafiки. Вона ініціює Pygame, налаштовує екран, відслідковує введення з клавіатури та змінює поведінку гри залежно від стану.

9.1 Ініціалізація

Ініціалізація: Створює вікно гри, запускає Pygame та задає заголовок. Також створює об'єкти екрана та годинника для контролю частоти кадрів.

```
1 def main():
2     try:
3         pygame.init()
4         screen = pygame.display.set_mode((TOTAL_WIDTH, HEIGHT))
5         pygame.display.set_caption("     i     ")
6         clock = pygame.time.Clock()
```

Лістинг 17: Ініціалізація головної функції

9.2 Початковий стан

Початковий стан: Ініціалізує змінні: стан гри (`"start"`), поле, фігури, позиції, рахунок, рівень тощо. Змінна `game_over` відслідковує завершення гри.

```
        game_state = "start"
        grid, current, next_shape, color, next_color, pos, score, level,
        lines_cleared, fall_timer, fall_delay = init_game()
        game_over = False
        running = True
```

Лістинг 18: Початковий стан гри

Головний ігровий цикл: Цикл працює, поки гравець не вийде. `dt` зберігає час між кадрами для керування затримкою падіння фігури.

```
        while running:
            dt = clock.tick(FPS)
```

Лістинг 19: Головний ігровий цикл

9.3 Обробка ігрового стану

Оновлення рівня та швидкості: Збільшує рівень кожні 10 ліній. Зменшує затримку `fall_delay`, щоб фігури падали швидше з підвищенням рівня.

```
            if game_state == "playing" and not game_over:
                fall_timer += dt
                level = lines_cleared // 10 + 1
                fall_delay = max(50, 600 - (level - 1) * 50)
```

Лістинг 20: Оновлення рівня та швидкості

9.4 Обробка подій

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
```

Лістинг 21: Обробка подій виходу

```
    elif event.type == pygame.KEYDOWN:
        if game_state == "start":
            if event.key == pygame.K_RETURN:
                game_state = "playing"
            elif event.key == pygame.K_ESCAPE:
                running = False
        elif game_state == "playing":
            if game_over:
                if event.key == pygame.K_r:
                    grid, current, next_shape, color, next_color, pos,
                    score, level, lines_cleared, fall_timer,
                    fall_delay = init_game()
                    game_over = False
                elif event.key == pygame.K_ESCAPE:
                    game_state = "start"
                    game_over = False
            else:
```

```
                new_pos = pos[:]
                new_shape = current
                if event.key == pygame.K_LEFT:
                    new_pos[0] -= 1
                elif event.key == pygame.K_RIGHT:
                    new_pos[0] += 1
                elif event.key == pygame.K_DOWN:
                    new_pos[1] += 1
                elif event.key == pygame.K_UP:
                    new_shape = rotate(current)
                elif event.key == pygame.K_SPACE:
                    #
                    while valid(grid, current, [pos[0], pos[1] + 1]):
                        pos[1] += 1
                    #

                elif event.key == pygame.K_ESCAPE:
                    game_state = "start"
                    grid, current, next_shape, color, next_color, pos,
                    score, level, lines_cleared, fall_timer,
                    fall_delay = init_game()
                if valid(grid, new_shape, new_pos):
                    pos, current = new_pos, new_shape
```

Лістинг 22: Обробка клавіш залежно від стану

Вихід із гри: Закриває гру при натисканні кнопки "Закрити вікно".
Керування клавіатурою:

- `ENTER` — початок гри
- `ESC` — вихід або повернення в меню
- `R` — рестарт гри після Game Over
- Стрілки — рух фігури (ліво, право, вниз)
- `UP` — обертання фігури
- `ПРОБІЛ` — швидке падіння фігури

9.5 Гравітація та падіння фігури

Автоматичне падіння фігури: Фігура поступово падає вниз відповідно до встановленої затримки. Якщо фігура більше не може рухатися — вона фіксується на полі, очищуються заповнені лінії, оновлюється рахунок та генерується наступна фігура. При неможливості розмістити нову фігуру встановлюється стан `game_over`.

```
if game_state == "playing" and not game_over and fall_timer >=
    fall_delay:
     pos[1] += 1
     if not valid(grid, current, pos):
         pos[1] -= 1
         for y in range(current.shape[0]):
             for x in range(current.shape[1]):
                 if current[y, x]:
                     grid[pos[1] + y, pos[0] + x] = color
         grid, cleared = clear_rows(grid)
```

```
        score += cleared * 100 * level
        lines_cleared += cleared
        current = next_shape
        next_shape = random.choice(shapes)
        color = next_color
        next_color = get_shape_index(next_shape) + 1
        pos = [COLS // 2 - current.shape[1] // 2, 0]
        if not valid(grid, current, pos):
            game_over = True
    fall_timer = 0
```

Лістинг 23: Автоматичне падіння фігури

9.6 Відображення екрана

Графічне відображення: Виводить стартовий екран, основний екран гри або екран завершення гри залежно від поточного стану `game_state`. Для стану Game Over додатково відображаються заблоковані фігури на полі та інформація про фінальний рахунок.

```
if game_state == "start":
    draw_start_screen(screen)
    pygame.display.flip()
elif game_state == "playing":
    if not game_over:
        draw(screen, grid, current, pos, color, next_shape,
             next_color, score, level)
    else:
        screen.fill(colors[0])
        for y in range(ROWS):
            for x in range(COLS):
                if grid[y, x]:
                    draw_block(screen, colors[grid[y, x]],
                               pygame.Rect(x * CELL, y * CELL, CELL, CELL
    ))
        draw_grid(screen)
        pygame.draw.rect(screen, (200, 200, 200),
                         (0, 0, WIDTH, HEIGHT), 3)
        draw_game_over(screen, score, level)
        pygame.display.flip()
```

Лістинг 24: Відображення різних станів гри

9.7 Завершення роботи

Обробка помилок: Якщо сталася помилка — вона виводиться в консоль разом з повним трасуванням стека, а гра закривається коректно з очищенням ресурсів Pygame. Умова `if __name__ == "__main__":` забезпечує запуск функції тільки при прямому виконанні файлу.

Функція `main()` об'єднує всі компоненти гри: логіку, графіку та введення, забезпечуючи безперервну та керовану гру з підтримкою пауз, рестарту та прогресії складності через систему рівнів.

```
except Exception as e:
    print(e)
    traceback.print_exc()
```

```
    pygame.quit()
    sys.exit()

pygame.quit()
sys.exit()

if __name__ == "__main__":
    main()
```

Лістинг 25: Обробка помилок та завершення

10 Висновки

Реалізація гри Тетріс на Python з використанням pygame та numpy демонструє ефективне поєднання графічних можливостей та числових обчислень. Структурований підхід до організації коду, використання констант, модульність функцій та чітке розділення логіки забезпечують створення стабільної та розширюваної гри.

Ключові переваги такого підходу:

- Чіткість та читабельність коду
- Легкість модифікації та розширення
- Ефективна робота з масивами через NumPy
- Плавна графіка завдяки pygame
- Модульна архітектура для простого супроводу