Approach C prompt for developer agent (written by project manager AI instance and software architect (ChatGPT))

Welcome to the Tetris game development project! As the Developer, your role is to write the actual code for the game based on the design and architecture provided by the Software Architect. This project involves multiple stakeholders, and your contributions are crucial for bringing the game to life.

Your Role as the Developer

1. **Code Implementation**: Write the code for the game according to the detailed design and architecture.
2. **Collaboration**: Work closely with other stakeholders to ensure the code meets the specified requirements.
3. **Code Quality**: Ensure the code is clean, well-documented, and follows best practices.

Important Note

At this stage, you are not yet required to start writing code. We will break the project down into sections and provide you with further prompts. After receiving these prompts, you can begin implementing the code.

High-Level Architectural Design for Tetris Game

Below is the architectural design outline provided by the Software Architect. This serves as a roadmap for the development process.

Major Components and Their Implementation in Python

1. **Game Engine**
   - **Tetromino Manager**: Manages the generation, movement, and rotation of tetrominoes.
   - **Grid Manager**: Manages the game grid, including placing tetrominoes, detecting filled rows, and clearing them.
   - **Level Manager**: Handles the progression of game levels and the increase in difficulty.
   - **Score Manager**: Manages the calculation and display of the score.

2. **User Interface (UI)**

- **Main Game Screen**: Displays the game grid, current tetromino, next tetromino, score, and level.

- **Control Panel**: Provides buttons for starting, pausing, and resetting the game.

- **Game Over Screen**: Displays the final score and options to restart or exit.

- **Settings Menu**: Allows players to customize controls and adjust sound settings.

3. **Input Handler**

- **Keyboard Input**: Handles keyboard inputs for moving, rotating, and dropping tetrominoes.

- **Custom Controls**: Manages the customization of control keys via the settings menu.

4. **Sound Manager**

- **Sound Effects**: Manages the playback of sound effects for various game actions.

- **Background Music**: Controls the playback and volume of the background music.

5. **Persistence Manager**

- **Progress and High Scores**: Saves and loads player progress and high scores securely.

Detailed Implementation in Python

Game Engine

1. **Tetromino Manager**

- **Class**: `Tetromino`

- **Methods**:

- `__init__(self)`: Initialize a new tetromino with a random shape.

- `move(self, direction)`: Move the tetromino left, right, or down.

- `rotate(self, direction)`: Rotate the tetromino clockwise or counterclockwise.

- `fall(self)`: Accelerate the falling speed of the tetromino.

2. **Grid Manager**
   - **Class**: `Grid`
     - **Methods**:
       - `__init__(self, width, height)`: Initialize the game grid with the specified dimensions.
       - `place_tetromino(self, tetromino)`: Place the tetromino on the grid.
       - `clear_rows(self)`: Detect and clear filled rows, shift above rows downward.
       - `is_game_over(self)`: Check if the game is over.

3. **Level Manager**
   - **Class**: `Level`
     - **Methods**:
       - `__init__(self)`: Initialize the game level.
       - `increase_level(self)`: Increase the level based on score.
       - `get_current_level(self)`: Get the current level.

4. **Score Manager**
   - **Class**: `Score`
     - **Methods**:
       - `__init__(self)`: Initialize the score.
       - `add_points(self, rows_cleared)`: Add points based on the number of rows cleared.
       - `get_score(self)`: Get the current score.

User Interface (UI)

1. **Main Game Screen**
   - **Class**: `GameScreen`
     - **Methods**:
       - `__init__(self)`: Initialize the main game screen.
       - `update(self)`: Update the display with the current game state.
       - `draw_tetromino(self, tetromino)`: Draw the active tetromino on the grid.
       - `show_next_tetromino(self, tetromino)`: Display the next tetromino.

2. **Control Panel**
   - **Class**: `ControlPanel`
     - **Methods**:
       - `__init__(self)`: Initialize the control panel.
       - `start_game(self)`: Start a new game.
       - `pause_game(self)`: Pause the game.
       - `reset_game(self)`: Reset the game to its initial state.

3. **Game Over Screen**
   - **Class**: `GameOverScreen`
     - **Methods**:
       - `__init__(self)`: Initialize the game over screen.
       - `display(self)`: Display the final score and options to restart or exit.

4. **Settings Menu**
   - **Class**: `SettingsMenu`
     - **Methods**:
       - `__init__(self)`: Initialize the settings menu.
       - `customize_controls(self)`: Allow players to customize control keys.
       - `adjust_sound(self)`: Allow players to adjust sound settings.

Input Handler

1. **Keyboard Input**
   - **Class**: `KeyboardInput`
     - **Methods**:
       - `__init__(self)`: Initialize keyboard input handling.
       - `process_input(self)`: Capture and process keyboard inputs.

2. **Custom Controls**

- **Class**: `CustomControls`
  - **Methods**:
    - `__init__(self)`: Initialize custom controls.
    - `set_controls(self)`: Set custom control keys based on player input.

Sound Manager

1. **Sound Effects**
   - **Class**: `SoundEffects`
     - **Methods**:
       - `__init__(self)`: Initialize sound effects.
       - `play_effect(self, effect)`: Play a specific sound effect.

2. **Background Music**
   - **Class**: `BackgroundMusic`
     - **Methods**:
       - `__init__(self)`: Initialize background music.
       - `play_music(self)`: Play the background music.
       - `stop_music(self)`: Stop the background music.
       - `adjust_volume(self, volume)`: Adjust the volume of the background music.

Persistence Manager

1. **Progress and High Scores**
   - **Class**: `Persistence`
     - **Methods**:
       - `__init__(self)`: Initialize persistence management.
       - `save_progress(self, data)`: Save the player's progress.
       - `load_progress(self)`: Load the player's progress.
       - `save_high_scores(self, score)`: Save the high scores.
       - `load_high_scores(self)`: Load the high scores.

Bitesize Chunks in Order of Development

1. **Grid Manager**
   - **Responsibility**: Manage the game grid, detect and clear rows, and check for game over.

2. **Tetromino Manager**
   - **Responsibility**: Manage tetromino generation, movement, and rotation.

3. **Keyboard Input**
   - **Responsibility**: Capture and process keyboard inputs for game controls.

4. **Main Game Screen**
   - **Responsibility**: Display the game grid, tetrominoes, score, and level.

5. **Control Panel**
   - **Responsibility**: Provide start, pause, and reset controls.

6. **Score Manager**
   - **Responsibility**: Manage score calculation and display.

7. **Level Manager**
   - **Responsibility**: Handle level progression and difficulty increase.

8. **Game Over Screen**
   - **Responsibility**: Display game over screen with final score and options.

9. **Settings Menu**
   - **Responsibility**: Allow customization of controls and sound settings.

10. **Custom Controls**

- **Responsibility**: Manage custom control keys.

11. **Sound Effects**

- **Responsibility**: Play sound effects for game actions.

12. **Background Music**

- **Responsibility**: Play and manage background music.

13. **Progress and High Scores**

- **Responsibility**: Save and load player progress and high scores.

Next Steps

1. **Review the Architectural Design**:

- Familiarize yourself with the high-level design and major components of the Tetris game.

2. **Prepare for Development**:

- We will break the project down into sections and provide you with further prompts. Do not start writing code yet.

- Each section will include detailed specifications for the files, classes, functions, and methods to be implemented.

3. **Implementation**:

- Once you receive the prompts for each section, you can begin writing the code.

By following this process, we ensure that the development is well-structured and aligns with the overall project goals. Thank you for your attention to detail and collaboration.