Final Project: INFO102

Quanzhi, Charles

Overview

Hello everyone! Welcome to the final project INFO102, which counts for the remaining 3 assignments. Over the past few weeks, you have learned a lot about programming, including Python programming, data structures, and software engineering skills such as abstraction and testing. Now it's time to apply those skills!

For this project, you will build your favorite game Genshin Impact in python, where you can explore and fight with your friends in a fantastic magical world!!!

Just kidding, it's your good old classic Tetris! In this project, you will develop the game from scratch using the most widely used object-oriented programming techniques. By the end, you will have a playable version of the game on your computer. If you're not familiar with Tetris, check out this super exciting video of the Tetris World Championship: https://youtu.be/L_UPHsGR6fM. Even if you already know what Tetris is, this video is still amazing!

Here is the grading overview for the project.

Grading Overview

- Basic Functionality (65%)
  - Minimal features of the Tetris game.
- Advanced Features (25%)
  - There are some additional features you may choose to add to your project. Feel free to select the ones you prefer. We will not provides you instructions in this part.
- Coding Qualities and Styles (10%)
  - A professional programmer does not settle for code that simply "runs". Code quality and style are also important. Your code should be properly commented, tested, and abstracted. A detailed rubric on coding styles and quality will be released later.

We understand that completing this task individually may be difficult, especially given that you have only been learning programming for several weeks. We will provide you with detailed instructions on basic functionalities, allowing you to quickly learn the basic principles of OOP and software development. With this knowledge, you will be equipped to freely develop advanced features.
Also, don't hesitate to reach out to us if you have any questions or encounter any difficulties. We are here to support you throughout the project and help you master the skills you need to succeed as a programmer.

Submission Instruction

There will be two checkpoints for this project.

- The first checkpoint, due on **April 27th**, is for basic functionality. To complete this checkpoint, you should
    1. zip all of your source code files and upload the ZIP file to the Assignments section of Sakai.
    2. submit a demo video, no longer than three minutes, that showcases the required features for basic functionality.
    3. include a simple documentations (5-6 lines are enough) on **how to use your program**, such as the key controls for rotation and movement.
- The second checkpoint, **due on May 7th**, is for advanced features. Similar to checkpoint 1, you should
    1. zip all your source codes and upload your ZIP file to the Assignments section of Sakai
    2. submit a demo that showcases how your advanced feature works.
    3. Include a simple documentation that **briefly** introduces the **implemented feature and how you implemented it**. The documentation should be about half page.
- **It is important to note that the length of the report will not affect your grade.**

Skeleton Code Structure

The skeleton code contains a key packages that you'll be using: `KBHit` provides some basic methods for getting inputs, as well as basic code structure for Tetris:

- `Block.py` - The blocks on the Tetris game.

- `Game.py` - The controller of the Tetris game.
- `Board.py` - The game board of the Tetris
- `test_Block.py` - A unit test sample for your development
- `main.py` - The main program

Keep reading for the detailed explanation of each file.

Basic Functionalities (65%)

In this project, we will show you how to develop software from scratch using the object-oriented programming (OOP) paradigm. Programming with OOP is very different from previously used procedure-oriented programming (POP), in which we use functions as the smallest units of operation. When programming with POP, we think about the order of different tasks. For example, to find two elements in a list whose sum equals a given value, we first iterate through all element pairs in the array, then check if the sum equals the given value, and finally return the result if the condition is true. In POP, we translate our tasks into a sequence of operations, which is then implemented by the programming language.

In OOP, we approach questions in a completely different manner. We no longer think (and should not think) about the procedures. Instead, we identify what objects exist inside our software, what features and behaviors they should have, and what the relationships are among the objects. This might be counterintuitive for beginners, but it has been proven to be one of the best ways to manage large software systems. Almost all software used today is developed using OOP. So, please take some time to convince yourself to "think about objects, not procedures." It is essential for developing other software systems in the future.

Repeat "Think about objects, not procedures" three times in your mind, and let's start our journey to Tetris.

Task 0 Tetris Structure (0%)

Let's consider what objects should be included in a Tetris game. First, we need a Block class to represent blocks in different shapes. In my design, a Block should have these attributes: the actual shape of the block, the position of the block, and the direction that denotes the current orientation of the block. With regard to behavior, a Block needs to be able to move down, left, and right, as well as rotate, and also should be able to return the current shape of the block.

Refer to the `Block.py` file for details and carefully read it to understand how class attributes and methods correspond to the design.

In addition, we need a Board to store information about the game. Take a moment to consider what attributes and behavior a Board should have if you were designing this class. In my design, a Board should have a nested list to store the game state, as well as a Block that is currently dropping down. The Board's behavior should include moving the Block down/left/right/rotate, deleting a line if it is full, generate a new block at the top of the board, checking if the user has lost the game, and checking if a move is valid (i.e., will the rotation cause the current shape to collide with already-existing bricks? Will a move take our shape outside the board? etc.).

Refer to the `Board.py` file for details and carefully read it to understand how class attributes and methods correspond to the design. Only a subset of the methods are declared in this file. You are going to implement more methods by yourself in future tasks.

You may think that the Board should also have some way to interact with the keyboard to control the game. Of course, you can do that, but we will not do it here. A basic principle of OOP design is the principle of single responsibility: a class should only do one thing. Our Board should only be responsible for the pure game state.

To control the game, we create a class called Game which is responsible for interacting with the user to control the game flow. First, a Game must include a Board (i.e., the game state). The Game should also include methods that can display the Board to the user and start the game.

Refer to the `Game.py` file for details and carefully read it to understand how class attributes and methods correspond to the design. Only a subset of the methods are declared in this file. You are going to implement more methods by yourself in future tasks.

Lastly, we need a main function to run our game.

Refer to the `main.py` file for details.

Congratulation! You finished task 0 of the project. Now you must have a big picture in your mind how our Tetris game will looks like. If you still find some points are unclear, go read the skeleton file provided. Do not proceed until you understand everything in the skeleton.

Task 1 Minimal End-to-End System (15%)

After have an initial idea on the Tetris. It's time for us to write some codes. But where shall we start?

The answer is: start from a minimal end-to-end system.

This principle in software engineering is called "Minimum Viable Product" or MVP. The idea is to start with the absolute minimum set of features that are required for a functional product, and then iterate and build upon that base.

To getting started, we are going to build a simplest (maybe boring) Tetris game. In which everything our program will do is just generate a block on the top and let it dropped on the bottom. This task, although simple, requires a proper implementation of methods in `Block`, `Board`, and `Game`.

Let's first think about what methods we need to implement.

- In `Block` we need at least `moveDown` for the block to move downward.
- In `Board` we need `tryMoveDown` to move the block downward, `isBlockValid` to validate the block's position ensure the block stop at the bottom, `dump` to write the information of block to the board once the block reach the bottom.
- In `Game` we need `display` to print the board information to the command line, `run` to move a block downward until it reach the bottom.

Hint: there is a method in `Board.py` called `toView()` Read the comment and the function body to find out what it does. It would be useful in implementing `display()`

Fill in the blank inside this methods and input the following command in command line, to see if your minimum system works properly.

```
python main.py
```

Kindly reminder: Write unit tests after you finish each methods above. It will reduce your time on debugging by a lot. You can find a example of unit testing using framework `unittest` in test_Block.py. You can run the test by typing the following command in the command line.

```
python test_Block.py
```

Please do not write all tests after you developed everything just for the style grade. I will be really upset if you do that.

Task 2 Add Control (20%)

In this task, you are going to write codes to interact with the user. Allowing user use keyboard to control the rotation and movements. Previously, we used `input()` to get users input. This function is not suitable for our situation. `input()` will blocking our program, wait until users hit ESCAPE. In our game, we do not want our program been blocked if no input provided.

Believe it or not, input/output (I/O) is actually a complex topic in computer science. While in Python all you need to do is use the `print()` or `input()` functions, a sequence of intricate operations is conducted by your computer to display messages on the screen or read inputs from the user. If you're curious about what happens under the hood, you may want to consider taking courses in operating systems or computer organization. However, this topic is beyond the scope of our project.

To help you deal with I/O issues, we provide an open-sourced library called `KBHit`. With this library, you can focus on implementing the functionalities of your Tetris game without worrying about the complexities of I/O.

You will mainly use two methods from `KBHit.py`, `kbhit()` and `getch()`

- `kbhit()` return true if any key on the keyboard was hit, return false elsewise. It will not block the program.
- `getch()` will read one character of the user’s input. This read operation will block the program. So only use `getch()` when `kbhit()` return true.

There is a sample code between line 115 to 129 of `KBHit.py` which demonstrate the basic usage of `KBHit.py`. You can use the following command to run the sample code. Play around with it to figure out how this library works.

```
python KBHit.py
```

It's time to take your Tetris game to the next level by adding controls. For this task, some functions have already been declared, and all you need to do is fill in the blanks. However, for other functions, you will need to implement them yourself. Additionally, some functions may need to be updated, such as `isBlockValid()`, which may require validation checks for vertical borders, collisions with other blocks, and more.

> Here's a helpful hint for you: you may want to include a delay before each downward movement, allowing the user time to input their desired action. The easiest way to implement this is to add a while loop before each downward movement and include `kbhit()` inside the loop.

Implement, test, and debug your functions. Then run your code to see if it works as your expectation.

Task 3 Update Board (10%)

In this task, you will finalize the last functionality of the game: updating the board after each turn. Once a block is dumped onto the board, before generating the next block, there are several things that need to be done.

- you need to remove any full rows by deleting them and moving all the blocks above the row downward by one position. If there are multiple full rows, they need to be removed all at once and the remaining rows moved downward accordingly.
- you need to check if there are any blocks above the upper threshold of the board. If there are blocks higher than the threshold, it indicates that the game is over.
- a new block should be generated for the next round.

Think about how to map these three tasks to class methods in our game. You may need multiple helper methods in different classes to make it work.

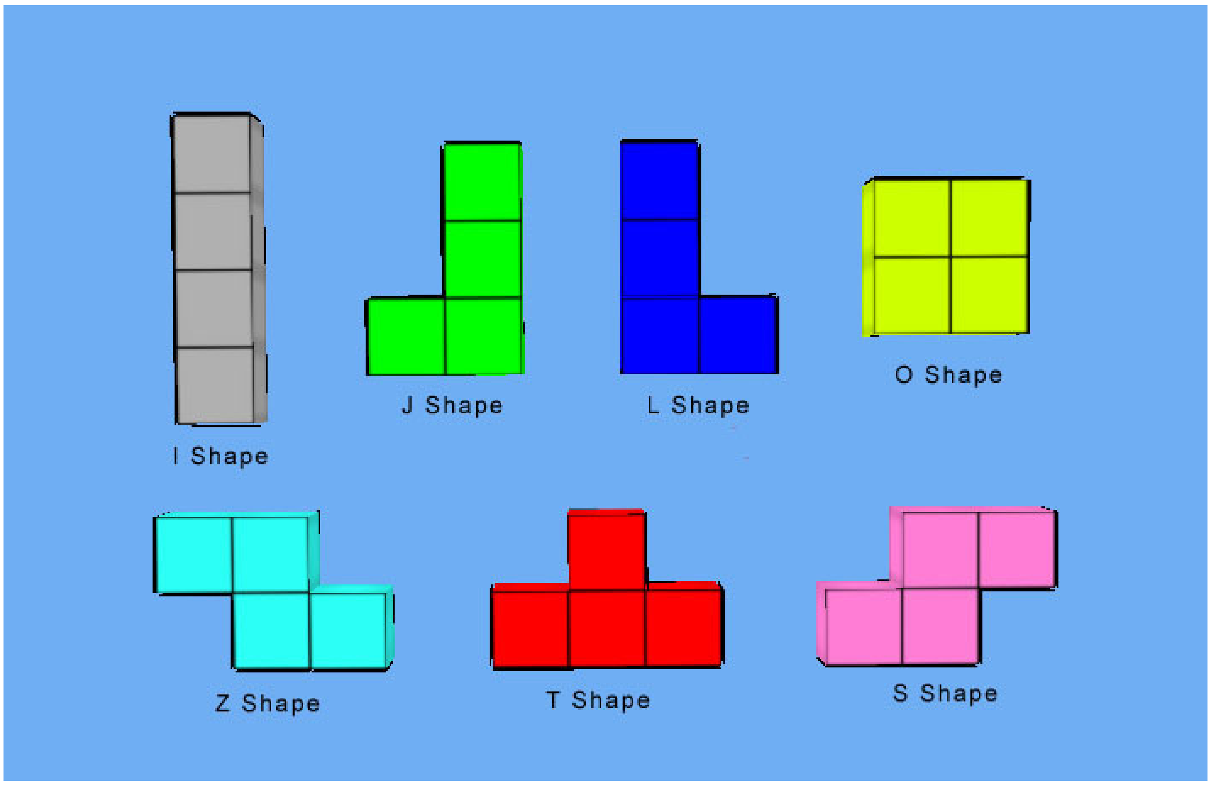
Additionally, if you haven't done so already, you need to add multiple rounds to your game. Your game should keep running until the player loses.

Implement, test, and debug your function. Play around with your game to see if there is anything unexpected.

Task 4 More Shapes (15%)

Now that you have a playable game, it's time to make it more interesting by adding more shapes. Currently, there is only one shape in our game, which can become quite boring after a while. In this task, you will be adding 5 different shapes to the game, designed according to your own preferences. However, at least 3 of the shapes should not be rectangular.

There are various ways to implement the new shapes, but for this project, you are required to use inheritance. You will need to modify the Block class and create new classes that inherit from it. If you implement inheritance correctly, you will only need to make a few updates in other places. This is the beauty of inheritance; if a block is a J shape block, it is still a Block, and all previous operations on Block should work properly on the J shape block.



The 7 shapes in classic Tetris, feel free to design your own shapes in the game. https://articles.informer.com/articles_uploads/0/842/a6d5d5ff85289b9eae2c0f93b115a63f_1_orig.jpg

Implement, test, and debug your functions. Play around with your Tetris and convince yourself the game is bug free.

```
python main.py
```

Congratulations! You have just built your very first game in Python. This is an incredible achievement that demonstrates your mastery of various skills in OOP. With this simple yet engaging game, you have gained hands-on experience in software development, including designing, testing, and refactoring.

Now, the time has come for you to unleash your creativity and add more features to your game. You are free to choose the tasks you would like to work on, but make sure that the summed weighted value of the tasks is equal to or greater than 25% to gain full marks. Additionally, remember to write tests for your advanced features to ensure that your program is working correctly.

If you have any other ideas for interesting features, feel free to reach out to us. We can evaluate the difficulty of your proposed features together and potentially incorporate them into your grade.

Advanced Features (25%)

Select tasks from the below advanced task list to gain this 25% of the grade.

1. **Implement different levels of difficulty:** Add multiple levels of difficulty with varying speeds and/or different starting configurations of blocks on the board (5%)
2. **Randomly Generated Blocks:** The block patterns should not remain the same. Make the block sequence different in each play (5%)
3. **Add special pieces with unique abilities:** Create special pieces that have unique abilities, such as clearing multiple rows at once or freezing blocks in place for a short period of time (15%)
4. **Implement a high score system:** Create a system for storing and displaying high scores.
    - level 1: record highest score in current play (5%)
    - level 2: record highest score of all past plays even after the user exit the game (10%)
5. **Add multiplayer support:** Implement multiplayer support, allowing multiple players to compete against each other in the same game. For example, the game should display two boards on the screen at same time and use different keys to control the boards.(25%)
6. **Add sound effects and music:** Incorporate sound effects and background music to enhance the game play experience (15%)
7. **Implement a pause and resume functionality:** Allow players to pause the game and resume playing later.
    - level 1: Pause and resume in current play (5%)
    - level 2: Allow player close the game and resume the play since last exit (15%)

8. **Add a preview of the next piece:** Show a preview of the next piece that will be dropping down on the board, giving the player more time to strategize (10%)
9. **Create different game modes:** Develop different game modes, such as a timed mode or a survival mode where the player has a limited number of moves to clear as many lines as possible (15%)
10. **Create a replay feature:** Allow players to replay their game after it has ended, so they can analyze their mistakes and improve their skills (25%)

PS: The Genshin Impact joke (which I hope was not too awkward!) was adapted from an assignment designed by Andrew Hilton - a truly talented and energetic professor at Duke ECE. The original version is “Your project this year is going to be your favorite game - Fortnite. Just kidding, it’s your good old classic TETRIS.”