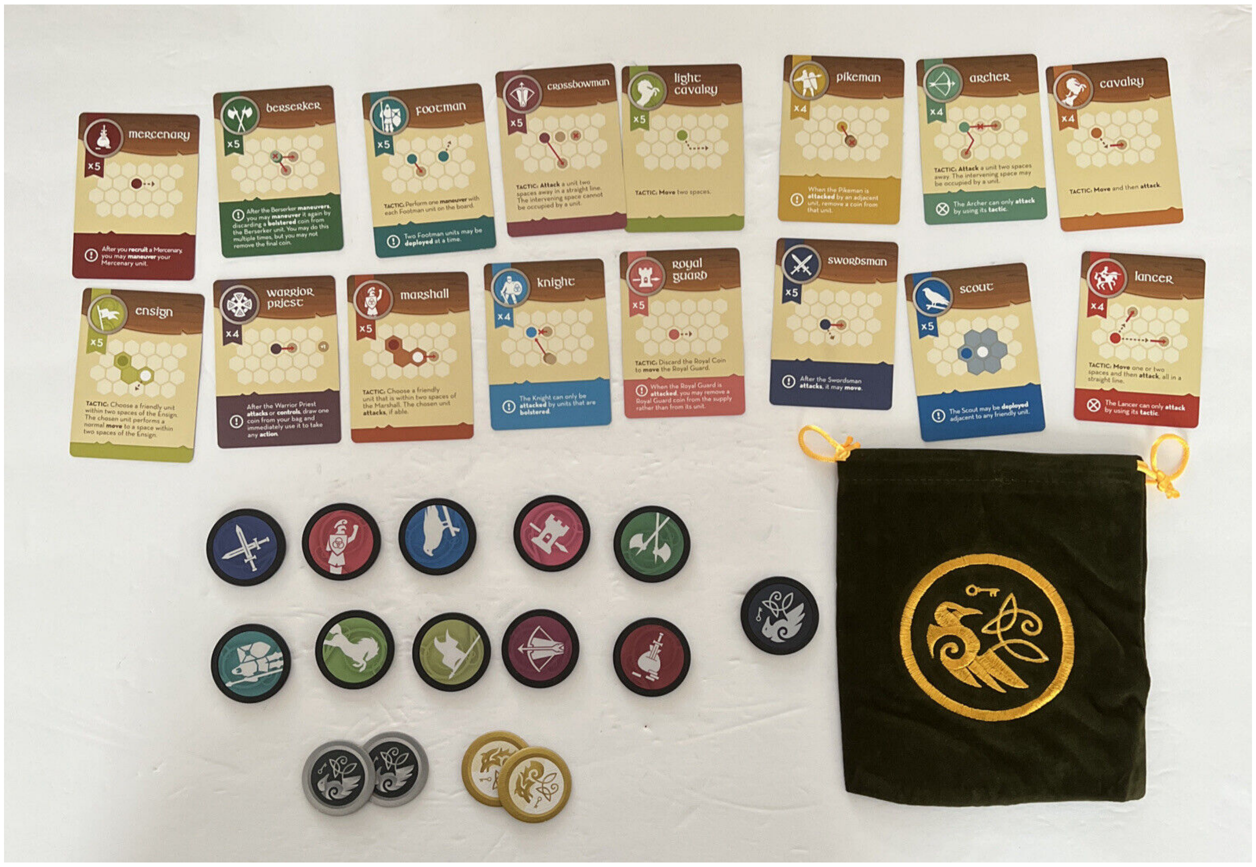
Ramón Íñiguez Bascuas 24/02/2023

EVENTBRITE

WARCHEST-LITE
GENERAL INFORMATION



eventbrite

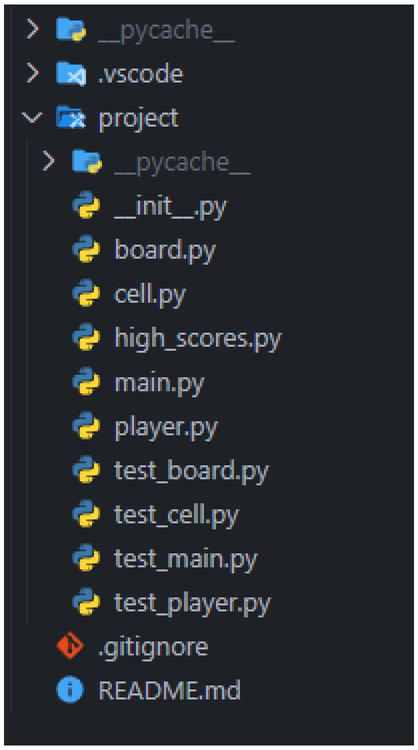
Index

Requirements

Since it is a relatively simple game in terms of external dependencies and importing language built-in functions, the only requirement is to run the **main.py** script using **Python 3.10+,** as one of the classes uses the newly added **match case** statement.

Project structure

The project structure is as defined in the following image:



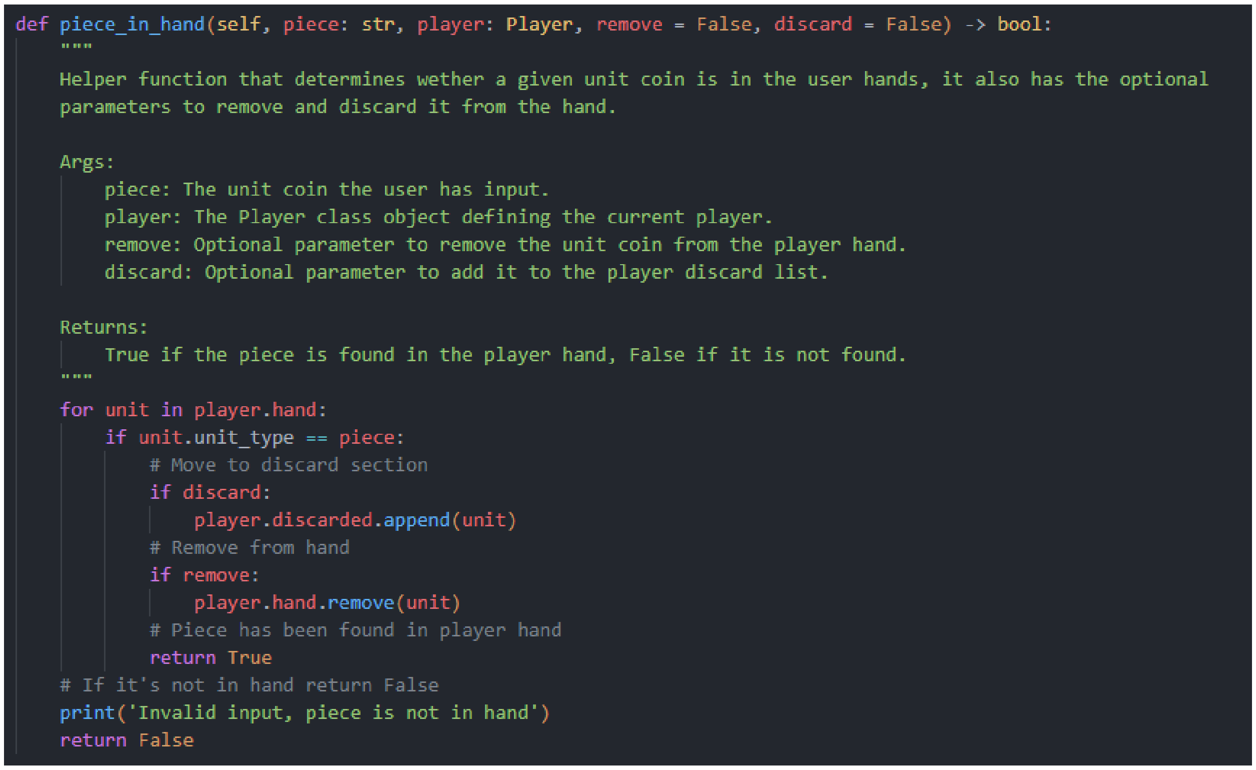
```
> __pycache__
> .vscode
v project
  > __pycache__
    __init__.py
    board.py
    cell.py
    high_scores.py
    main.py
    player.py
    test_board.py
    test_cell.py
    test_main.py
    test_player.py
  .gitignore
  README.md
```

The reason for this is that it is a simple project in terms of files, and if we added any new folders such as 'tests' to group all the tests it would then mean following the Python sys.path convention, which ultimately makes everything more complex and my objective is to keep things as simple as possible.

Therefore, it follows a project structure similarly to how it is done in the Go language, where we group every file and its respective test file in the same folder, allowing for easy imports between the different files where necessary. If the project was more complex and there was a larger amount of files then it would be necessary to group everything in a different way, with different packages using the __init__.py in each.

Thought process

Every function and every class in the project has its respective docstring, which follows the Google format, as well as comments in various lines to explain the current flow as seen in the following image:



```python
def piece_in_hand(self, piece: str, player: Player, remove = False, discard = False) -> bool:
    """
    Helper function that determines wether a given unit coin is in the user hands, it also has the optional
    parameters to remove and discard it from the hand.

    Args:
        piece: The unit coin the user has input.
        player: The Player class object defining the current player.
        remove: Optional parameter to remove the unit coin from the player hand.
        discard: Optional parameter to add it to the player discard list.

    Returns:
        True if the piece is found in the player hand, False if it is not found.
    """
    for unit in player.hand:
        if unit.unit_type == piece:
            # Move to discard section
            if discard:
                player.discarded.append(unit)
            # Remove from hand
            if remove:
                player.hand.remove(unit)
            # Piece has been found in player hand
            return True
    # If it's not in hand return False
    print('Invalid input, piece is not in hand')
    return False
```

This means that the actual documentation of each class and the thought process is included in the code base. One of the things I do want to comment though is that I have followed the logic the video demo has shown, where you introduce the from coordinate, the piece to move and the position to move, but this method ultimately makes the error checking more complex as it would be easier to just prompt the user to input the from coordinate and based on that only allow the player to move the unit that is contained there, by checking for errors once, such as: empty cell, invalid cell, piece not belonging to player or if the piece us in the hand, and not how it is currently done where you have to then ask the user which piece to move, and check for errors at every step.

Additionally I tested the most important functions in each file/class, handling the most complex logic to showcase my testing skills, but following the skeleton I have created the rest of the functions can also be tested.

It is also important to note that the game is highly robust to user input errors and will not crash in most scenarios, as making it fully robust would make it too complex & long and the way it has been implemented allows for the correct testing of each function.