Scrabble Code Report

Joel Morley*
Physics Department, University of Manchester.
(Dated: May 24, 2021)

Scrabble board game was written in C++. Most of the rules where implemented except the ability to draw new tiles from the bag and blank spaces. These could be implemented with more time. Play ends naturally using the pass limit. Multiple languages was thought of and with a good file system could be implemented.

I. INTRODUCTION

The board game Scrabble was chosen. This involves multiple players placing letter tiles on a board, building words and scoring points based on a number of criteria. The full rules can be found online[1].

II. CODE DESIGN AND IMPLEMENTATION

The project was built around the `scrabble` class. This contains the game loop as well as holding and processing all other resource's used. The resources involved use the classes `dictionary`, `player`, `bag` and `board`.

The `scrabble` class and all resources that cannot be determined at run-time can be (de)serialized through the `serializable` interface:

```
class serializable
{
    public:
    virtual std::ostream& serialize(std::ostream& stream) const = 0;
    virtual std::istream& deserialize(std::istream& stream) = 0;
};
```

Each implementation is unique for the inheriting class but they all can take the stream, place/remove their data into/from it and return it back to the caller. This was used in favour of overloading the << and >> operators as it improved readability due to the fact both `draw` and `serialize` have the same signature.

A. Game Loop

The game loop is within the `scrabble` class calling two internal methods:

```
while (running){
    this -> draw(std::cout);
    this -> on_command(std::cin);
};
```

The `draw` method is from the `render` interface which is implemented in all classes that are required to render to the screen during game play:

```
class render
{
    public:
    virtual std::ostream& draw(std::ostream& stream) const = 0;
};
```

Each implementation is unique for the inheriting class but they all take the output stream, render to it and return it back to the caller.

The `on_command` method takes a line from the user and try's to call the appropriate function for the string entered:

- `help`: Calls `help` method which sets the message value to a help message explaining the tiles and commands.
- `save [file_name]`: Calls `save` method which serializes the class to the provided `file_name` file in `/data/save` directory. If no `file_name` is provided the date and time is used.
- `exit`: Calls `save` method and sets `running` to false, exiting the game loop.
- `set <x> <y> <character>`: Calls `set` method which attempts to set the provided parameter's in the current board.
- `reset`: Calls `reset` method which reverts all `set` calls this turn.
- `turn`: Calls `turn` method which checks whether the board is valid, then calculates score and then switches to the next player.
- `pass`: Calls `pass` method which passes the turn to the next player. If a game is passed 6 times the game is over.

If the command fails to run or an invalid command is entered an error response is placed in the `message` string which is displayed in the next `scrabble::draw` call.

B. dictionary

Implements the `serializable` interface. It is used to check whether a provided word is contained in the internal dictionary. A cache is used to store words that

* Student Code: 10306710

have already been found to reduce the amount of times the full dictionary has to be searched. Words are stored in a `set`, each as a lower case `string`. A `set` was used as it guarantees objects are unique and sorts the words allowing for a quick `find` call when checking whether the word is present.

The dictionary was taken from a website online and can be switched out with other language dictionaries if required [2].

C. player

Implements the `serializable` and `render` interfaces. It is used to store information about a player. Namely their name, score and hand. The name is stored as a `string` and is provided during construction. The score is a `double` and is altered by the scrabble class during scoring and winning. The hand is a vector of type `letter` that the `scrabble` class manages. The max hand size and number of players are also stored in this class. The `player` class includes a number of helper functions to support handling of it's hand:

- `has_letter`: Returns a `bool` stating whether the hand contains a specific `char`.
- `get_letter_position`: Returns the first index of a specific `char`.
- `has_room_in_hand`: Returns a `bool` stating whether there are any empty spots in the hand.
- `add_letter`: Attempts to add the provided `letter` pointer to the hand. Returns a `bool` indicating whether it was successful.
- `remove_letter`: Deletes the letter at the provided index. Returns a `bool` indicating whether it was successful.

D. bag

Implements the `serializable` interface. It is used to store letters that are to be used later in the game. `random_letter` method populates a provided `letter` pointer with a random letter from its storage as well as returning a `bool` representing whether it was a success(if there was enough letters in the bag). `fill_hand` method takes a player pointer and attempts to (re)fill their hand with random letters.

E. board

Implements the `serializable` and `draw` interfaces. Contains the letters and tiles. The letters are stored as a `vector` of `letter` while the tiles are stored as a `vector` of `piece`. The `piece` interface is used to enforce the implementation of the `serializable` and `draw` interfaces as well as providing methods to distinguish between the concrete types.

```
class piece: public render, public serializable
{
    public:
    enum type{
        LETTER, BOOST, SYMBOL
    };

    static piece::type get_type(char charector);
    static char get_charector(piece::type type);

    virtual piece::type get_type() const = 0;
};
```

This distinction allows the `piece_factory` helper class to construct pieces of the appropriate type.

The board is also managed by the `scrabble` class and as such contains helper methods:

- `get_horizontal_word`: Returns the horizontal word at the provided x and y index's as a string.
- `get_verticle_word`: Returns the vertical word at the provided x and y index's as a string.
- `get_horizontal_word_positions`: Returns the horizontal word at the provided x and y index's as a deque of pairs for the index's of the letters of the word.
- `get_verticle_word_positions`: Returns the vertical word at the provided x and y index's as a deque of pairs for the index's of the letters of the word.

F. main

The main class is used to set up the game at first. This asks the user whether they want to load or start a new game as well as providing a way to pick the number of players. In future it would also include a file searching system to allow the selection of separate bags, dictionaries and boards. The players names could also be changed in the future.

III. RESULTS

The game is run by inputting commands to alter the game state accessing the functions laid out in section II A. The game can be saved during play or during exit. It can be loaded during startup in the menu section. Different boards can be produced and used. Only one (the standard board) is currently present in the game.

The first turn is special in the fact that it forces the player to play in the centre and also allows for infinite passing of turns. From there its standard play. There is currently no automatic win detection but the play will end naturally once the pass limit is reached.

Two of the rules where not implemented. This was the one where you can switch out your hand with pieces in the bag. This would not be too hard to implement, time constraint meant that there was not time to fully test it. Similarly blank pieces were not implemented for the same reason.

IV. DISCUSSION AND CONCLUSION

The game has been built with multiple languages in mind but was never fully realised due to the lack of finding a good file explorer that works cross platform. Once the C++17 standard becomes more accepted in compilers the filesystem class could be used to list and access not just new languages but also save files. Multiple languages would also require having a language translation for each piece of text in game, this would require a significant alteration to the scrabble::draw command.

The full rules could be implemented if more time was given to test them. A way of detecting the end game could also be produced.

[1] Hasbro, Scrabble rules: Official word game rules: Board games (2020).

[2] Dictionaries, *JUST WORDS!*. Accessed 24 May 2021. [Online].