ScrabbleCzeker

Project Documentation

***Development Head**:*

Maksymilian Słowiński

***Developers**:*

Stanisław Wasik

Szymon Jędrzejewski

Dawid Białek

https://github.com/st-wasik/ScrabbleChecker/

Poznan University of Technology - Faculty of Electrical Engineering

Poznań 2019

Abstract **5**

Project overview **5**
GitHub Repository 5
Motivation 5
Overview 6
Distribution of work 6
Main features 6
How to run 7
How to use 7
Polish characters encoding note **7**

Project design **8**
Used technologies 8
Image Processing Module (Python) 8
Words validation module (Haskell) 8
Result Display Module (C++) 9
Modules connection 9

Modular structure **10**
Image Processing Module (Py) 10
Module overview 10
Environment configuration 10
Data flow 11
Input 11
Main flow 11
Output 13
Sources documentation 13
Capture 13
show_ip_webcam() 13
board_detection_BRISK(board_img) 14
draw_grid(warped_board) 14
detect_tiles(warped_board) 14
Letters 14
matrix_match(matrix) 14
tesseract_recognition(name, thresh, blur) 15
Interesting problems and solutions 15
Low quality web cam 15
Light intensity 15
Words Validation Module (Hs) 16
Module overview 16
Environment configuration 16
Build 16
build.sh 16
build-clean.sh 17
Data flow 17

Correct Words Dictionary 17
Input 17
Main data flow algorithm 18
Words processing algorithm 18
Points calculation algorithm 19
Output 19
Processed board: 20
Points: 20
Sources documentation 20
Main 20
main :: IO () 20
beginNextCheck :: Set [Char] -> [(Int, Int)] -> IO () 21
processWords :: Set [Char] -> [(Int, Int)] -> String -> IO () 21
initializeDictionary :: IO (Set String) 21
filterDictionaryWords :: [String] -> [String] 21
printOut :: Show a => a -> IO () 21
printOutPoints :: [(a,b)] -> IO () 21
STest 21
ex1 :: [Char] 22
SData.SChar 22
data SValid 22
Valid 22
Invalid 22
NotSet 22
PartOfInvalid 22
SingleLetter 22
ConnectionPoint 22
NotConnected 22
Eq SValid 23
Show SValid 23
data SChar 23
SChar :: Char -> SValid -> (Int, Int) -> SChar 23
Show SChar 23
type SWord = [SChar] 23
empty :: (Int, Int) -> SChar 23
word :: SWord -> String 23
printSCharList :: [SChar] -> IO () 23
SData.SBoard 24
fromList :: [Char] -> Matrix SChar 24
empty :: Matrix SChar 24
updateNotConnectedWords :: Matrix SChar -> Matrix SChar 24
placeSChars :: Matrix SChar -> [SChar] -> Matrix SChar 24
allWords :: Matrix SChar -> [SWord] 24
sWords :: [SChar] -> [SWord] 24
setProperValidness :: SChar -> Matrix SChar -> Matrix SChar 25
SAlgorithm.SPoints 25

wordValue :: [(Int, Int)] -> SWord -> Int 25
charValue :: [(Int, Int)] -> SChar -> Int 25
charPoints :: SChar -> Int 25
charPosFactor :: SChar -> Int 25
charPosWordFactor :: [(Int, Int)] -> SChar -> Int 26
SAlgorithm.SValidatior 26
processScrabbleBoard :: Matrix SChar -> S.Set 26
-> [(Int, Int)] 26
-> (Matrix SChar, [(String, Int)], [(Int, Int)]) 26
calculatePoints :: Matrix SChar -> [SWord] -> [(Int, Int)] 26
-> [(String, Int)] 26
updateSValid :: ([SWord], [SWord], [SWord])
-> ([SWord], [SWord], [SWord]) 27
validatedWords :: Matrix SChar -> S.Set String 27
-> ([SWord], [SWord], [SWord]) 27
Result Display Module (C++) 28
Module overview 28
Data flow 28
Input 28
Main flow 28
Output 29
Sources documentation 30
Application 30
void addTile(int letter, int x, int y, char status) 31
std::vector<sf::RectangleShape> buildBoard() 31
void clear() 31
int convertCharacter(int value) 31
std::shared_ptr<tgui::ListBox> createList() 31
void read_stream() 32
void read_words(); 32
void run() 32

Abstract

Main task of this program is to help Scrabble players. This program has to detect words placed on the board, check if words are correct and calculate points for each word. At the end virtual board is displayed with hints about words correctness and corresponding to each word points.

Attached words dictionary is prepared for Polish. There is a possibility to run program for other languages, but code contains **special** facilities for polish letters. Also implemented Scrabble rules are taken from polish version of Scrabble.

If you want to run this program for other languages you have to change dictionary file and points calculation rules implemented in Words Validation Module.

Project overview

GitHub Repository

The whole project is located under following URL:

https://github.com/st-wasik/ScrabbleChecker

Motivation

This project was developed, because we wanted to develop project with image and text processing. We also wanted to improve programming skills in Python and Haskell and use newly acquired skills.

We also found that Checkers Checker will be not so advanced in development and boring.

Overview

This solution is build with three independent modules, to make work comfortable to all developers. All three modules are developed in this project. Additionally this project use extern application for Android named IP Webcam. For more refer to Image Processing module description.

Unfortunately development version of ScrabbleChecker is provided for Windows only, because Display Module was created using Visual Studio. You can try at your own to port this module to Linux.

To make development easier for each team member we found that every developer will choose technology that in his option is best for assigned task.

Distribution of work

Stanisław Wasik (Haskell)	Dawid Białek (Python)	Maksymilian Słowiński (Python)	Szymon Jędrzejewski (C++)
Verification of revied words	Board detection and transformation	Recognition of letters	Displaying board and words
Calculating points for words	Detecting tiles on board and cutting them out	Sending detected letter to validation module	Displaying list of words with points

Main features

This project offers only two features:

1. Checking if words placed on the Scrabble board are correct.
    a. Application detects invalid words and displays information which letters are part of incorrect words. It also displays information which letters are

not connected to other words and which letters are part of correct and incorrect word.

2. Calculating points according to each correct word.
    a. All points are calculated using polish version of Scrabble rules.

How to run

To run this project you must meet all requirements listed in every module *Build* chapter. First you need to compile Validation and Display modules.
If the compilation finished without any errors you may attempt to run ScrabbleChecker, but before you can run this program you have to change few lines in python code:

- In file **capture.py** in line 243 and 244 you have to change IP address to the IP of your phone.
- In file **letters.py** in line 45 you have to change path to **tesseract.exe.**

After these change you can run this program.

To do this simply run development script provided with project repository. Script name is ***run.bat***.

How to use

1. Launch a IP Webcam app on your phone and start streaming.
2. Start ScrabbleChecker by staring **run.bat** script.
3. When you see a preview of phone camera point it at game board and hit *Spacebar,* then wait for the outcome.
4. Repeat taking the photos at end of every turn of player.

Polish characters encoding note

We came across character encoding **problems** in **all** project modules. Each module used other encoding so we decided to apply CP1250 encoding to command line and files. It serves to properly handle polish letters like ‘ą’, ‘ę’, ‘ó’.

Unfortunately module written in Haskell applies Unicode encoding for all characters. The best result we obtained when to console input stream was applied CP1250 encoding and dictionary file was read with UTF-8 encoding. For other encodings the dictionary did not work properly, there was problem with comparing characters incoming from input with the characters placed in dictionary.

Project design

Used technologies

Following technologies are used for listed modules:

Image Processing Module (Python)

This module is written in Python.

Python was chosen, because the easy of use and multitude of solutions that are based on OpenCV python.

To recognize letters we use Tesseract OCR, because it can identify polish characters with default datasets and it’s popularity gives lot of sources for finding information.

Words validation module (Haskell)

This module is written in Haskell.

Haskell was chosen, because data processing in declarative languages is very simply. High abstraction of functional programming provides lucidity and brevity of code. Additionally there are much high-order functions in Haskell that can make text processing quick and comfortable.

This module uses only standard library *Prelude* and two additional modules, that can be installed using default Haskell tools, like *cabal* or *stack*.

Result Display Module (C++)

This module is written in C++.

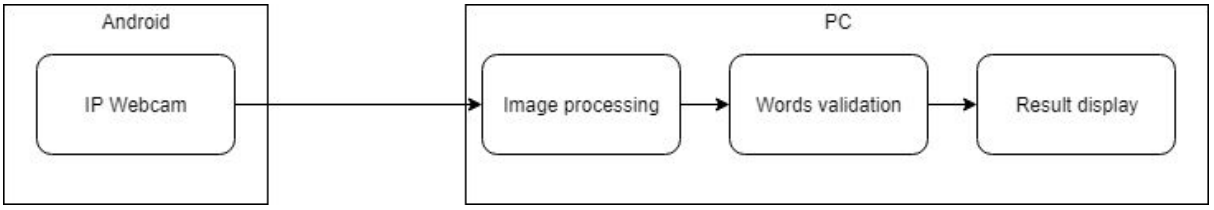
C++ was chosen, because it is compatible with SFML and TGUI libraries. Access to mentioned libraries provides easy and fast way to create simple, working graphic user interface.

SFML was chosen, because it is more friendly for begginers than OpenGL. Additionally it has a very good source documentation and support.

TGUI was chosen, because it provides ready solutions of creating widgets. To implement fully working widget with the help of TGUI library, all you need is a feed lines of code. It also provides easy callback system.

Modules connection

This diagram presents data flow between modules in ScrabbleChecker.



IP Webcam provides image captured from camera. This image is transferred to PC and processed by Image Processing Module. Next detected letters are transferred to Words Validation Module. Then lists with information about correctness and points is transferred to Result Display Module. This module displays Scrabble board, presents placed words and calculated points.

Connection between Android and PC is realized by WiFi . WiFi access point is created by smartphone. Connection between modules in PC is realized with console pipeline.

Next chapter contains detailed information about modules structure.

Modular structure

Image Processing Module (Py)

Module overview

This module name is CheckerEyes. It was created to detect the game board and letters placed on it and recognize what are the letters.

Communication with Words Validation Module is established by standard IO. Incoming and outcoming data format is described in Data Flow chapter.

Environment configuration

CheckerEyes module is interpreted with Python 3.7.3.
For module to work you need to install some additional libraries:

- OpenCV (opencv-python)
- Pillow (Pillow)
- Python-tesseract (pytesseract)
- Numpy (numpy)
- Matplotlib (matplotlib)

Best way to install this libraries and all dependencies is trough ***pip*** tool.
Additionally it uses Tesseract which has to be installed separately.

Data flow

Input

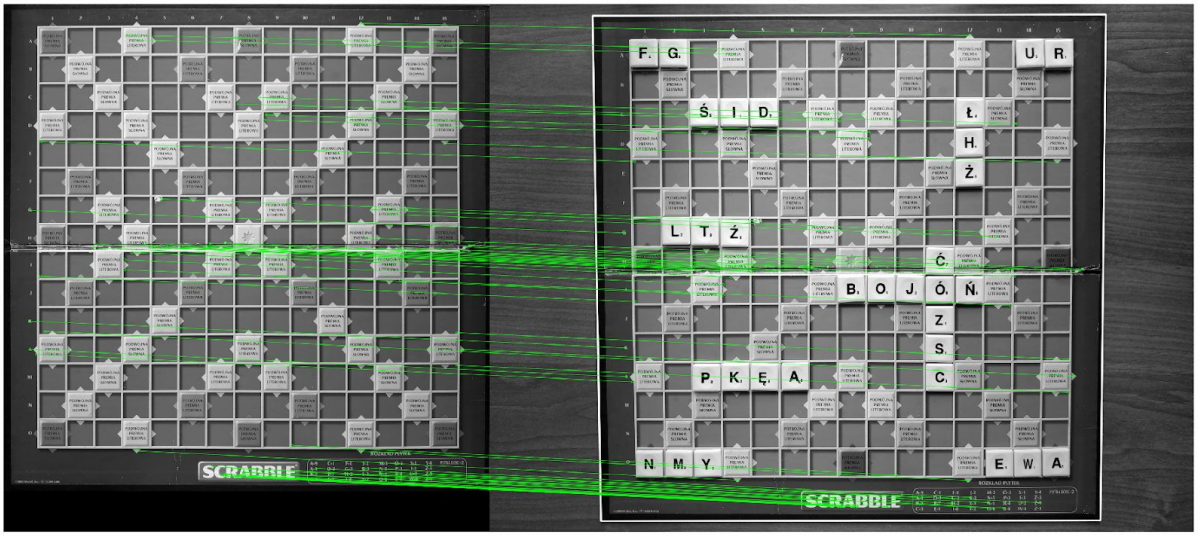
Data that enters this module are images taken with smartphone camera and sended with use of application *IP webcam* over WiFi and save as png file for use.

Main flow

1. Get image for processing
2. Find board on image and transform it to match reference image
3. Lay grid on to transformed picture
4. Cut out field using grid and check if it contains tile with a letter
5. Add to list 0 if there is no letter or cropped image is there is one
6. Repeat checking tiles (steps 3 and 4) until all are checked
7. Check if element of list is a number or something else
8. If there is number add space to output string
9. If there is image process it with Tesseract and add result to output sting
10. Repeat for all elements in list (steps 7 and 8 or 9)
11. Print output string
12. Wait for next image

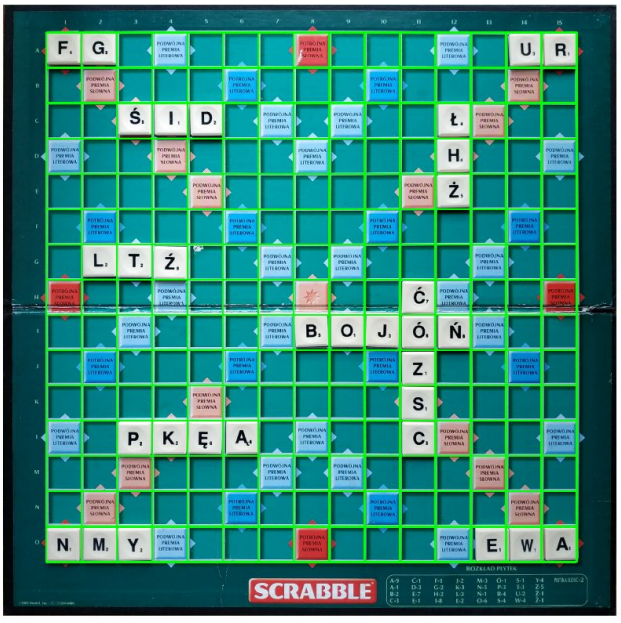
Matching image of board to the reference. Green lines shows connections between keypoints. This image has only almost parallel lines, which shows good correlation between points of interest.

Matching



Board image after warping and drawing grid. Warping the image allows a program to successfully detect fields and draw lines. Then each field can be cropped into separate image for further processing.

Warped board



Output

Processed data is outputted to Validation module as string which represents flattened matrix of the board with indexes of places on board looking like this:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165
166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225

In this string squares that don't have any letters are represented as spaces and letters as uppercase contrapart.

Sources documentation

This chapter provides description of all available functions and data types.
Documentation structure is divided for module source files.

Capture

```
show_ip_webcam()
```

> Connects to the smartphone camera using ip address shared by IP Web Cam application. After successful connection, function displays live feed from the camera. When spacebar key is pressed, function will send photo request to the camera and after receiving response it will save the photo and send it to the board detection function.

```
board_detection_BRISK(board_img)
```

Takes board photo as an argument, changes board and reference image color formats from RGB to BGR and scales them down if needed. Then it creates keypoints in both photos and matches them. After choosing the best matches it performs perspective transformation on the board photo, using homography.

```
draw_grid(warped_board)
```

Draws grid on the image of warped board.
Returns image with green grid.

```
detect_tiles(warped_board)
```

Takes warped board and separates fields into individual images. Then after converting them from BGR to HSV color format, it detects which photo shows tile with a letter.
Returns array of tiles with letters and their position on the board.

Letters

Contains all functions use to recognize letters on tiles

```
matrix_match(matrix)
```

Takes list of numbers and images, for images calls **`tesseract_recognition,`** for numbers adds space to output string
Ends in printing out output string for Validation Module

```
tesseract_recognition(name, thresh, blur)
```

Takes image of letters, finds what letter is that.

Returns uppercase character of that letters.

Interesting problems and solutions

Low quality web cam

One of the problems we had was a low quality webcam. With it we had only around 34x34 pixels of tiles images. That was not enough.
To solve this problem we exchange a normal webcam with a smartphone. With use of app called *IP webcam* we could take high-quality pictures and operate on them. We got something around 4x the quality using this method and this resolved bulk of ours problems and gave us a option to use some kind of optical character recognition (tesseract in ours case).

Light intensity

There is a problem with color of light and light intensity.
This problem can’t be easily solved with programing measures, because it is more of the hardware problem. There are some way to mitigate this. One is to change the threshold of tile detection. We don't have any specific values, because it can be very different even, if light condition change very little. Second and most important method to mitigate this problem is to have soft and not to bright light on board and to avoid reflex of light on board.

Words Validation Module (Hs)

Module overview

This module name is CheckerService. It was created to validate words and count points.

Communication between Image Processing Module and Result Display Module is established by standard IO. Incoming and outcoming data format is described in Data Flow chapter.

This module is responsible for whole control of Scrabble game rules.

Environment configuration

CheckerService module was compiled using ***ghc*** compiler, version 8.4.3. Haskell version is Haskell2010.

To build CheckerService project required are additional libraries:

- matrix,
- containers

Dependencies can be installed using ***cabal*** tool.

Other build information are located in:

- CheckerService.cabal,
- stack.yaml

Build

Build scripts are located in CheckerService directory.

---

build.sh

```
ghc src/Main.hs -o CheckerService -isrc -O2
-fdiagnostics-color=always -Wdeprecations
```

This script builds project using second optimisation level. Additionally provides warnings about deprecations.

---

build-clean.sh

```
ghc src/Main.hs -o CheckerService -isrc -O2 -no-keep-hi-files
-no-keep-o-files -fdiagnostics-color=always -Wdeprecations
```

This script provides the same functionality as ***build.sh*** but deletes also files created during build: *.hi, *.o.

Data flow

Correct Words Dictionary

Words list was obtained from https://sjp.pl/slownik/growy/. This is list of correct words used in games like Scrabble. List contains polish words.

In the program the dictionary is represented as Set. The implementation of Set is based on size balanced binary trees. This cause that for about 3 million of words the Set has only 22 levels. It means that when program checks if word is member of the dictionary, it must compare only 22 words.

Unfortunately building set from words list take some time because of balancing the underlying tree.

Input

Data incoming from Camera Module contains only letters ordered in fifteen rows of fifteen letters. All rows are concatenated in single string. This module process such formatted input string.

The order of letters in the string (according to the shape of game board) is presented below.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165
166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225

Main data flow algorithm

1. Read dictionary-words from file.
2. Create set from words.
3. Fetch from input representation of detected Scrabble Board (from Python module),
    a. close application if line is equal to “exit”.
4. Process words (in a separated algorithm).
5. Print processed matrix as single row string.
6. Print list of tuples of words and corresponding words.
7. Update information about used fields for future points count.

8. Fetch next data from input (point 3).

Words processing algorithm

1. Create from input string matrix that represents game board.
2. Process board:
    a. detect all words placed on the board,
        i. if it is not a word, rather a single letter, place it into single letters list.
    b. check if every word is member of dictionary set,
        i. if is, place it into valid words list,
        ii. if not, place it into invalid words list.
    c. Mark every letter of the word with proper information about word's correctness.
    d. Create new board representation.
    e. Place single letters on new board (place every letter on its position).
    f. Place correct words on new board.
    g. Place incorrect words on new board.
    h. Check if every word has a connection point with other word or it is placed on the central tile of the board (Scrabble rules).
    i. Mark words with the connection points.
    j. Calculate points for created words (separate algorithm).
    k. Return new board, list of tuples of words and points and a list of fields on which was placed words (information for points calculation algorithm).

Points calculation algorithm

1. If points for given words were calculated in previous turn,
    a. Return 0 points.
2. Calculate bonus factor for whole word,
    a. Map letters of given words to its positions,
        i. Map every position to bonus factor,
        ii. If field was used before change this field factor to 1,

iii. Return product from obtained factors.

3. Calculate points for every letter,
    a. Map letter to its position,
        i. Map every position to point value,
        ii. Map every position to letter bonus factor,
        iii. Multiply letter points by field bonus,
    b. Return sum of calculated points.
4. Return summed points multiplied by whole word bonus.

Output

Processed board is transmitted to Display Module, to display calculated result on the screen. Also calculated points are sent by standard output to display “words history” and obtained points. This data is send in two separated lines that can be distinguished by Display Module.

The format of output is shown below.

Processed board:

[Lc,Lc,Lc,Lc,...Lc]

where **L** is letter upper case and **c** is information about correctness.

The relevant letter and correctness codes are placed in SChar module. The orders letter in the output string is the same as in the input string.

Points:

[(word,points)]

In normal case the list contains only one tuple of word and points, but there can be more tuples e.g. ***[(word,points),(word,points)]***.

Sources documentation

This chapter provides description of all available functions and data types.
Documentation structure is divided for module source files.

Main

contains processing algorithm entry point.

```
main :: IO ()
```

Module entry point.

```
beginNextCheck :: Set [Char] -> [(Int, Int)] -> IO ()
```

Takes dictionary and list of used fields positions.

Gets input data and dispatches data flow to proper way, e.g. closes application if input string is equal to “exit”.

```
processWords :: Set [Char] -> [(Int, Int)] -> String -> IO ()
```

Takes dictionary, used fields positions list and input string.

Calls data processing algorithm and prints out its result.

```
initializeDictionary :: IO (Set String)
```

Initializes directory for use.

```
filterDictionaryWords :: [String] -> [String]
```

Drops words that are one letter long or have two the same letters.

```
printOut :: Show a => a -> IO ()
```

Prints immediately argument to stdout.

```
printOutPoints :: [(a,b)] -> IO ()
```

Formats and prints given points list to the stdout.

STest

contains data for testing processing algorithm correctness.

```
ex1 :: [Char]
```

Example data Char list to be loaded by SBoard constructor.
STest file contains several test lists, where first is `ex1`, second `ex2` etc.
Last test list is `ex12`.

SData.SChar

provides data types and functions to create and handle letters representation.

```
data SValid
```

This data type defines correctness level of letter.

Constructors:

```
    Valid
    Invalid
    NotSet
    PartOfInvalid
    SingleLetter
    ConnectionPoint
    NotConnected
```

- Valid - letter is part of correct word
- Invalid - letter is part of incorrect word
- NotSet - letter is empty (contains space character)
- PartOfInvalid - letter is part of both correct and incorrect words
- SingleLetter - letter isn't part of any word
- ConnectionPoint - letter is part of two correct words
- NotConnected - letter is part of word, that is not connected to board center tile or any other word

Instances:

```
    Eq SValid
    Show SValid
```

```
data SChar
```

Constructor:

```
    SChar :: Char -> SValid -> (Int, Int) -> SChar
```

Constructs SChar from letter, information about validness and position.

Instances:

```
    Show SChar
```

```
type SWord = [SChar]
```

Alias for list of SChar.

```
empty :: (Int, Int) -> SChar
```

Constructs empty SChar from its position. Fills SChar with empty (space) character.

```
word :: SWord -> String
```

Converts SWord to String.

```
printSCharList :: [SChar] -> IO ()
```

Prints formated SChar list.

SData.SBoard

provides data types and functions to create and handle Scrabble Board representation.

```
fromList :: [Char] -> Matrix SChar
```

Creates Scrabble Board representation from Char list.

```
empty :: Matrix SChar
```

Creates empty Scrabble Board representation.
Uses empty characters (space) to fill created board.

```
updateNotConnectedWords :: Matrix SChar -> Matrix SChar
```

Marks words that are not connected with others or with board center.

```
placeSChars :: Matrix SChar -> [SChar] -> Matrix SChar
```

Updates SBoard with SChars. Places given SChars on SBoard taking into account already placed SChars and updates its validness.

```
allWords :: Matrix SChar -> [SWord]
```

Returns list of all words detected on the board.

```
sWords :: [SChar] -> [SWord]
```

This function is similar to Data.List.words but accept list of type SChar.
Takes list of characters and returns list of words.
This function is used by `allWords` function.

```
setProperValidness :: SChar -> Matrix SChar -> Matrix SChar
```

Sets proper validness for given SChar and places it on the SBoard
Takes into account already placed SChars and updates its validness.

SAlgorithm.SPoints

provides algorithm used to calculate points that Scrabble Player should get after placing word on the board.

```
wordValue :: [(Int, Int)] -> SWord -> Int
```

Takes list of used already used fields and word for which will be calculated points.
Returns points calculated for given word.

```
charValue :: [(Int, Int)] -> SChar -> Int
```

Takes list of used already used fields and character for which will be calculated points.
Calculates points taking into account letter value and additional field bonus.
Returns calculated points.

```
charPoints :: SChar -> Int
```

Takes character.
Returns value of character based on Scrabble rules.

```
charPosFactor :: SChar -> Int
```

Takes character.
Returns additional field position bonus based on Scrabble board and rules.
This bonus concerns single letter.

```
charPosWordFactor :: [(Int, Int)] -> SChar -> Int
```

Takes list of used already used fields and character for which will be calculated points.
Returns additional field position bonus based on Scrabble board and rules.
This bonus concerns whole word.

SAlgorithm.SValidatior

provides algorithm used to validate words placed on the board.

```
processScrabbleBoard :: Matrix SChar -> S.Set
    -> [(Int, Int)]
    -> (Matrix SChar, [(String, Int)], [(Int, Int)])
```

Processes SBoard:

1. Validates word with dictionary.
2. Mark SChars with proper info about validness.
3. Place SChars on SBoard.
4. Update SBoard with info about not connected words
5. Count points if all words are valid

```
calculatePoints :: Matrix SChar -> [SWord] -> [(Int, Int)]
   -> [(String, Int)]
```

Calculates points only if all words are correct, otherwise returns empty list. It means, that “words history” in the Display Module wont be updated if there is any wrong word.

```
updateSValid :: ([SWord], [SWord], [SWord])
   -> ([SWord], [SWord], [SWord])
```

Takes triple of validated words lists.
Updates SChars with info about validness.

```
validatedWords :: Matrix SChar -> S.Set String
   -> ([SWord], [SWord], [SWord])
```

Returns triple of words lists (Valid, Invalid, SingleLetters) validated using dictionary.

Result Display Module (C++)

Module overview

This module name is ScrabbleCzeker. It was created to display board with currently present tiles as well as list of words together with points for each word.

Communication with Words Validation Module is established by standard IO. Incoming and outcoming data format is described in Data Flow chapter.

Data flow

Input

Data that enters this module are pairs of arrays. First array represents fields on board together with tiles on the board. Second array represents pair of word and points, if word added to the board is correct, otherwise second array is empty.

Data entry is located in separate thread, to avoid main window freeze, if no data is incoming. Getting data from std::cin is blocking function, so it is better to use it in separate thread.

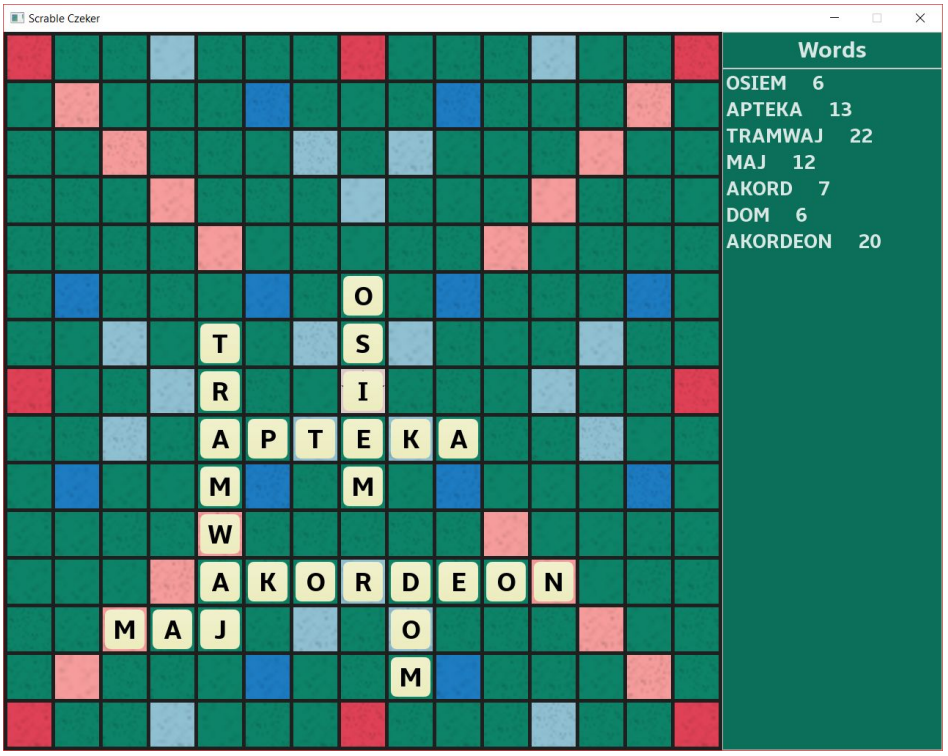
Main flow

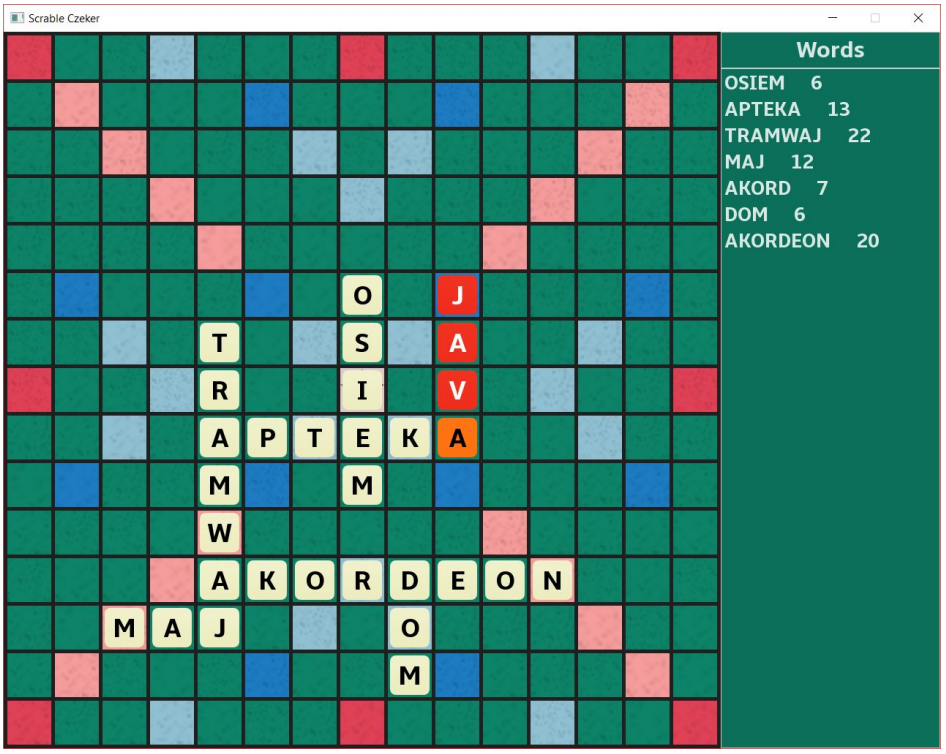
1. At the start buildBoard() is called to create empty board that will serve as background.
2. Calling createList() to create empty list on the right side of the board.
3. Data is transferred through stream to module.

4. Clear() is called inside read_stream() to clear board of tiles.
5. Stream is being read one character at the time.
6. If there is character present in polish alphabet, character is sent to convertCharacter() function to convert CP1250 encoding to Unicode needed for TGUI library.
7. Result of convertCharacter() function is sent, together with next character, that represents tile status, to addTile() function.
8. Tile is added on top of the board with coordinates based on its place in the stream.
9. Calling read_words() to pull words and points out of second array with the help of regex.
10. Concatenation of word and points is added to the list.
11. Application is waiting for next stream to be sent. After stream is sent, procedure is restarted from point 3, until exit signal is sent.

Output

Processed data is outputted to screen as image which represents physical board that looks like this:





Squares with letters have tiles that represents tiles on physical board. Empty squares don't have tiles.

The tiles are outputted in one of three colors:

- Beige-yellow - color of tiles, that indicates that word is valid,
- Orange - color of tiles, that indicates that the title is part of valid and invalid word at the same time,
- Red - color of tiles, that indicates that word is invalid.

Next to board is displayed list with words and points for each word.

Sources documentation

This chapter provides description of all available functions and data types.
Documentation structure is divided for module source files.

Application

Contains all functions present in Application

```
void addTile(int letter, int x, int y, char status)
```

Takes char converted to int, coordinates on the board (x and y) and status of the tile to create new object using Button class from TGUI library.

```
std::vector<sf::RectangleShape> buildBoard()
```

Builds board of 15x15 size. Types (colors) of squares are initialized from two-dimensional array that holds values corresponding to different squares on physical board (bonus and normal squares).

Returns vector of elements that need to be drawn with every loop.

```
void clear()
```

Clears tiles before next part of data from stream is read.

```
int convertCharacter(int value)
```

Converts characters from 1250 encoding to Unicode needed in TGUI library to display polish letters.

Returns integer value of characters sent to function.

```
std::shared_ptr<tgui::ListBox> createList()
```

Creates list used to store words and points from second array of data. No items are added at the start.

Returns pointer to list.

```
void read_stream()
```

Reads stream sent by Words Validation Module one character at the time. Ignores characters that separate data from 2 different squares. Reads only first array of data. Calls read_words().

```
void read_words();
```

Reads stream and matches stream parts to regex that separates words and points. Word and points are then concatenated and added to the list (initialized in createList()).

```
void run()
```

Main function of Application class. Calls buildBoard() and createList(). Creates separate thread that reads stream (function read_stream()). Handles events and is responsible for drawing objects created with help of SFML and TGUI libraries.
