NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

School of Electrical Engineering and Computer Sciences

CS250 DATA STRUCTURES AND ALGORITHMS

Project:

Scrabble Game

Submitted By:

Muhammad Hamza (173088)

Misbah Hamid (173211)

Rabbiya Qaiser (188070)

(BEE-8A)

Submitted To:

Sir Abid Rauf

Date of Submission:

04 Jan, 2018

NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY, SEECS

BONAFIDE CERTIFICATE

Certified that this project titled **"SCRABBLE GAME"**

is the bonafide work of

Muhammad Hamza

Misbah Hamid

Rabbiya Qaiser

of Class BEE-8A who carried out the project work under my supervision.

Sir Abid Rauf
Visiting Faculty
NUST, SEECS.

Dated:
04/01/2018

ABSTRACT

*This report outlines the design and development of a computerized **Two Player Scrabble Game**. Scrabble is the most popular word game ever published. The program has been written in C++ using Data Structures and Algorithms to run under Windows Operating System In addition, we have used Simple and Fast Multimedia Library(SFML)in order to generate graphics of the display of our game. The players score points in Scrabble by creating words with letter tiles on a game board. Each tile is assigned a numerical value, and as each new word is formed or each previously played word is modified, a score is recorded. At the end of the game, when all the tiles have been drawn and one player uses all his tiles, the game ends. Whoever has the highest number of points wins the game. But there is more to the game than an expansive vocabulary. An effective player should also be able to quickly find words in a jumble of letters. Developing this skill will not only improve your game, it will change the way you use your brain. The project provides a successful implementation of all basic rules and regulations of the scrabble game and also provides and checks all sorts of restrictions and limitations in order to prevent the players from breaking any sort of rules. Our project uses a subset of the dictionary containing twenty thousand words. The primary focus of the project is to prevent manual errors and cheating during the game and precisely checks and matches every word from the dictionary so that only valid words are made. The project thus provides a successful implementation of computerized Two Player Scrabble Game.*

Table of Contents

Abstract.....2

Progress Report.....5

Introduction.....5

Introduction to Scrabble 5

History.....6

The Scrabble Board 6

Extra Point Values.....7

Total Number of Scrabble Tiles.....8

Scrabble tiles Numerical Values.....8

Rules and Regulations.....10

Introduction to SFML.....13

Detailed Explanation of Window Module of Multi-Media In SFML.....14

Opening and managing a SFML window.....14

Windows Class Reference.....16

Events 19

Keyboard & Mouse.....21

Detailed Explanation of Graphics Module of Multi-Media In SFML 21

Drawing 2D Files.....23

Sprites and Textures.....25

Texts and Font.....27

Shapes.....28

Transforming Entities.....30

Data Structures and Algorithms.....31

Arrays.....31

Tries.....31

Outline Solution **33**

FlowCharts 34

Testing and Validation **53**

Problems Specification 53

Logical Errors 53

Future Developments **53**

Conclusion **54**

Preloaded Data **55**

Console Output **57**

Appendix **58**

References 67

PROGRESS REPORT

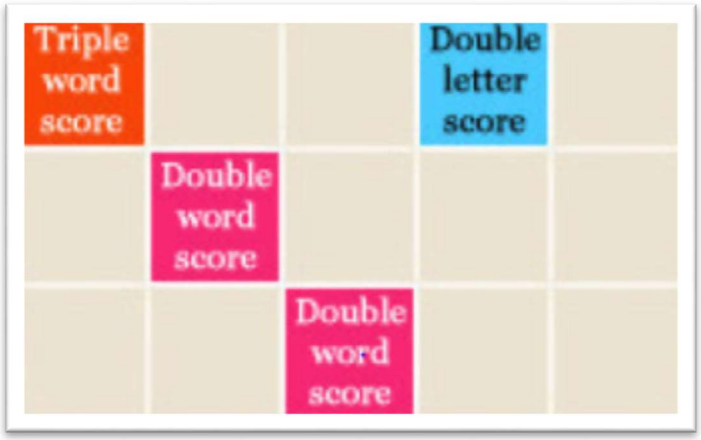
After the selection of our respective project i.e. Computerized Two Player Scrabble Game we as a team of three collectively studied all important features as well as rules and regulations of the game. The methods we employed for this purpose including browsing through the internet, by playing the scrabble game online as well as on the hard board to understand it completely, cumulative discussions and sharing and amendment of ideas presented by one another. These ideas included number of tiles, frequency and score of each tile, minimum letter words that can be made, order in which the tiles can be placed on the board, the initialization of the game etc. All these things were added step by step in the code which was started by learning and implementation of SFML Graphics Library. Further functions and implementations were made by making classes and sprites. At every step our code passed through a rigorous manual testing process for systematic as well as logical errors which were removed side by side. New ideas were added and implemented continuously as they were brought up. The ideas of done button, score board and pixels display were added to the code as soon as they were thought of by any group member. Through this process of step by step addition of new elements for the enhancement of code we came to the final product which is hereby presented in the form of our Computerized Two Player Scrabble Game.

INTRODUCTION

Introduction to Scrabble:

Scrabble is one of the most popular word come board game ever published that not only tests your skills to form words out of finite letter and polishes your voacublary but it also challenges your brain to solve problems in any given condition. The players score points in Scrabble by creating words on the tiles on the game board with the 7 letters provided to them on the rack on a purely random basis out of the total 100 tiles in the bag. Different types of tiles on game board carry different points. Each tile is assigned a numerical value, and as each new word is formed or each previously played word is modified, a score is recorded. The players would want to form as many high-scoring words as they can so they do not end up playing

low-scoring words when other possibilities exist. The tiles in the rack are replaced from the bag after each players move. At the end of the game, when all the tiles have been drawn and one of the player uses all his tiles, the game ends. Whoever has the highest number of points wins the game.

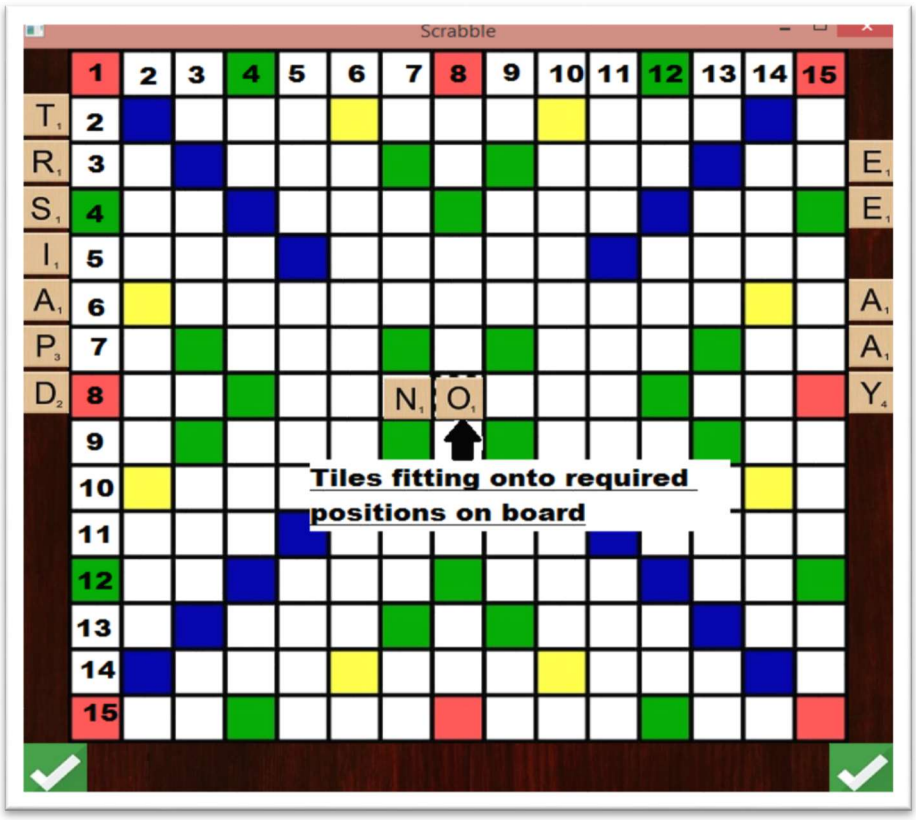


History:

Scrabble was invented in 1938 by an architect named Alfred Mosher Butts.He worked out how many points should be given for each letter by looking at books and newspapers and counting how often particular letters are used in the English language. He called the game "Criss-Crosswords". However, he was not successful in selling the game. In 1948 a lawyer called James Brunot bought the rights to make to game. Brunot made some of the rules easier, and changed the name to "Scrabble", which is a real word meaning: “to scratch around frantically”. Soon other firms around the world were buying rights to make the game. It had become so popular by 1984 that it was made into a daytime TV show on NBC television. The name Scrabble today is a trademark of Hasbro, Inc. in the US and Canada and of J. W. Spear & Sons PLC in other countries.

The Scrabble Board:

A standard Scrabble board consists of cells that are located in a large square grid. The board offers **15 cells high and 15 cells wide**. The **tiles used on the game will fit in each cell** on the board.



Extra Point Values:

When looking at the board, players will see that some squares offer multipliers. Should a tile be placed on these squares, the value of the tile will be multiplied by 2x or 3x. Some squares will also multiply the total value of the word and not just the single point value of one tile.

- **Double Letter Scores** - The light blue cells in the board are isolated and when these are used, they will double the value of the tile placed on that square.
- **Triple Letter Score** - The dark blue cell in the board will be worth triple the amount, so any tile placed here will earn more points.
- **Double Word Score** - When a cell is light red in colour, it is a double word cell and these run diagonally on the board, towards the four corners. When a word is placed on these squares, the entire value of the word will be doubled.
- **Triple Word Score** - The dark red square is where the high points can be earned as this will triple the word score. Placing any word on these squares

will boost points drastically. These are found on all four sides of the board and are equidistant from the corners.

- **One Single Use** - When using the extra point squares on the board, they can only be used one time. If a player places a word here, it cannot be used as a multiplier by placing another word on the same square.

Extra Point Values (Premium Words)	Position on Board
Double Letters	3, 11, 45, 165, 59, 179, 52, 36, 38, 108, 92, 122, 172, 186, 188, 116, 102, 132, 96, 98, 126, 128
Triple Letters	20, 24, 76, 136, 200, 204, 88, 148, 80, 84, 140, 144
Double Words	16, 32, 48, 64, 28, 42, 56, 70, 112, 196, 182, 168, 154, 208, 192, 176, 160
Triple Words	0, 7, 14, 105, 119, 210, 217, 224

Total Number of Scrabble Tiles:

There are exactly 100 tiles in Scrabble, but the distribution of the tiles is not even. There are more tiles for vowels than for most of the consonants. This distribution information comes in handy when you know that only one tile exists for each of J, K, Q, X and Z in the game. Those particular tiles are assigned high numerical values and provide the best opportunities for high scores. As the game progresses, you may be able to discern which opponent is holding a high-value tile and play to thwart its use.

Scrabble Tiles Numerical Values:

The total point value of all 100 tiles is 187. Although all tiles have a value, some are worth much more than others. In Scrabble, you want to form as many high-scoring

words as you can instead of playing low-scoring words when other possibilities exist. The numerical frequency and values of the tiles are distributed as follows:

List of Tables:

Tile	Frequency	Score
A	9	1
B	2	3
C	2	3
D	4	2
E	12	1
F	2	4
G	3	2
H	2	4
I	9	1
J	1	8
K	1	5
L	4	1
M	2	3
N	6	1
O	8	1
P	2	3
Q	1	10
R	6	1
S	4	1
T	6	1
U	4	1
V	2	4
W	2	4
X	1	8
Y	2	4
Z	1	10
Blank/Wild Tiles	2	0

Rules and Regulations:

- **Starting the Game:**
  - Without looking at any of the tiles in the bag, players will take one tile. The player that has the letter that is closest to "A" will begin the game.
  - A blank tile will win the start of the game.
  - The tiles are them replaced to the bag and used in the remainder of the game.
  - Every player will start their turn by drawing seven tiles from the Scrabble bag.
  - There are three options during any turn:
    - The players can place a word
    - They can exchange tiles for new tiles or they can choose to pass.
    - In most cases, players will try to place a word as the other two options will result in no score.
  - When a player chooses to exchange tiles, they can choose to exchange one or all of the tiles they currently hold. After tiles are exchanged, the turn is over and players will have to wait until their next turn to place a word on the board.
  - Players can choose to pass at any time. They will forfeit that turn and hope to be able to play the next time. If any player passes two times in a row, the game will end and the one with the highest score will win.

- **The First Word Score:**
  - When the game begins, the first player will place their word on the checkboard tile in the centre of the board.
  - The checkboard is a double square and will offer a double word score. All players following will build their words off of this word, extending the game to other squares on the board.
  - Play continues in a clockwise direction around the Scrabble board

- **Replacing Scrabble Tiles:**
  - Once tiles are played on the board, players will draw new tiles to replace those.
  - Players will always have seven tiles during the game.
  - Drawing tiles is always done without looking into the bag so that the letters are always unknown.

- **The End of Scrabble Game:**
  - Once all tiles are gone from the bag and a single player has placed all of their tiles, the game will end and the player with the highest score wins.

- **Tallying the Scrabble Scores:**
  - When the game ends, each player will count all points that are remaining on their tiles that have not been played. This amount will be deducted from the final score.
  - An added bonus is awarded to the player that ended the game and has no remaining tiles. The tile values of all remaining players will be added to the score of the player who is out of tiles to produce the final score for the game.
  - The Scrabble player with the highest score after all final scores are tallied wins.

- **Accepted Scrabble Words:**
  - All words in the agreed-upon dictionary, including inflections, are acceptable.
  - There are some words that are not allowed to be played and these include suffixes, prefixes and abbreviations.
  - Any word that requires the use of a hyphen or apostrophe cannot be played in the game. Any word that required the use of a capital letter is not allowed.
  - When playing an English version of the game, foreign words are not allowed to be placed on the board.

- **Tips for Beginners at Scrabble:**

  - Master the 101 two-letter words you can make in Scrabble.
  - Move on to the three-letter words and learn many of those, particularly the ones with high-scoring letters.
  - Be ready with words that use the Z, Q or K if you draw that tile.
  - Watch for opportunities to add a letter (or letters) to opponent's word to expand it and count both his tile points and your own.
  - Play defensively by not playing a word that can be easily modified by another player. The opponent will get your points and his own if he is able to add even a single letter to your word to form a new word.

Introduction to SFML:

SFML stands for ***Simple and Fast Multimedia Library.*** SFML is a cross-platform **software development library** designed to provide a simple application programming interface (API) to various multimedia components in computers. It is a simple to use and portable API written in C++. It is like an Object-Oriented Specification and Description Language (SDL). You can use SFML as a minimalist window system in order to use OpenGL, or as a complete multimedia library full of features to build video games or multimedia software's. SFML is:

- ***Multi-media:***
  SFML provides a simple interface to the various components of your personal computer, to ease the development of games and multimedia applications. It is composed of five modules in order to be as useful as possible:
  1. System provided by *SFML-system*
  2. Window provided by *SFML-window* (will be explained in detail)
  3. Graphics provided by *SFML-graphics* (will be explained in detail)
  4. Audio provided by *SFML-audio*
  5. Network provided by *SFML-network*

- ***Multi-platform:***
  With SFML, your application can compile and run out of the box on the most common operating systems: Windows, Linux, Mac OS X and soon Android & iOS.

- ***Multi-language:***
  SFML has official bindings for the C and .Net languages. And thanks to its active community, it is also available in many other languages such as Java, Ruby, Python, Go, and more.

Detailed Explanation of Window Module of Multi-Media In SFML

1) Opening and managing a SFML window:

Introduction:

- ***Opening a window:***

  Windows in SFML are defined by the sf::Window class. A window can be created and opened directly upon construction.

```
void main()
{

    RenderWindow window(VideoMode(670, 690), "Scrabble", Style::Default);
}
```

The first argument, the *video mode*, defines the size of the window (the inner size, without the title bar and borders). Here, we create a window with a size of 670x690 pixels.The sf::VideoMode class has some interesting static functions to get the desktop resolution, or the list of valid video modes for full screen mode. The second argument is simply the title of the window. This constructor accepts a third optional argument: a style, which allows you to choose which decorations and features you want. You can use any combination of the following styles:

Styles	Description
sf::Style::None	No decoration at all (useful for splash screens, for example); this style cannot be combined with others
sf::Style::Titlebar	The window has a Titlebar
sf::Style::Resize	The window can be resized and has a maximize button
sf::Style::Close	The window has a close button

sf::Style::Fullscreen	The window is shown in fullscreen mode; this style cannot be combined with others, and requires a valid video mode
sf::Style::Default	The default style, which is a shortcut for Titlebar | Resize | Close

- ***Bringing a window to life:***

If we do not implement any piece of code after our above implementation the window remains dead. Thus, we add further code in order to move, resize or close the window.

```
while (window.isOpen())
{
    Event event; //everything responsible for
    Vector2i mouseCoordInt;
    //new pos of tile vector
    int pos; //psoition of tile

    //shit done during game
    while (window.pollEvent(event))
    {
        if (event.type == Event::Closed)
            window.close(); }
```

The above code will open a window, and terminate when the user closes it.

First, we added a loop that ensures that the application will be refreshed/updated until the window is closed. Most (if not all) SFML programs will have this kind of loop, sometimes called the *main loop* or *game loop*.

Then, the first thing that we want to do inside our game loop is check for any events that occurred. Note that we use a while loop so that all pending events are processed in case there were several.

The pollEvent function returns true if an event was pending, or false if there was none. Whenever we get an event, we must check its type (window closed? key pressed? mouse moved? joystick connected? ...), and react accordingly if we are interested in it. In this case, we only care about the Event::Closed event, which is triggered when the user wants to close the window.

At this point, the window is still open and we have to close it explicitly with the close function. This enables you to do something before the window is closed, such as saving the current state of the application, or displaying a message.

After the window has been closed, the main loop exits and the program terminates.

- ***Playing with the window:***

  Basic window operations such as changing the size, position, title or icon are supported, but unlike dedicated GUI libraries, SFML doesn't provide advanced features. SFML windows are only meant to provide an environment for SFML drawing.

```
tableLayout.setPosition(Vector2f(35, 0)); //setting scrabble board
backgroundlayout.setPosition(Vector2f(0, 0)); //setting background
DoneButtonLayout1.setPosition(Vector2f(0, 600));
DoneButtonLayout2.setPosition(620, 600);
```

2)Window Class Reference:

sf::Window is the main class of the Window module.

The sf::Window class provides a simple interface for manipulating the window: move, resize, show/hide, control mouse cursor, etc. It also provides event handling through its pollEvent() and waitEvent() functions.

```
RenderWindow window(VideoMode(670, 690), "Scrabble", Style::Default);
```

Member Functions:

- **sf::Window::Window()**

  Default constructor. This constructor doesn't actually create the window, use the other constructors or call create() to do so.

- **Construct a new window:**

```
RenderWindow window(VideoMode(670, 690), "Scrabble", Style::Default);
```

  This constructor creates the window with the size and pixel depth defined in *mode*. An optional style can be passed to customize the look and behavior of the window (borders, title bar, resizable, closable, ...). If *style* contains Style::Fullscreen, then *mode* must be a valid video mode.

- **void sf::Window::close()**

  Close the window and destroy all the attached resources.

  After calling this function, the sf::Window instance remains valid and you can call create() to recreate the window. All other functions such as pollEvent() or display() will still work and will have no effect on closed windows.

- **void sf::Window::display()**

  Display on screen what has been rendered to the window so far.

- **void sf::Window::setPosition(const Vector2i &position)**

  Change the position of the window on screen.

  This function only works for top-level windows (i.e. it will be ignored for windows created from the handle of a child window/control).

  Parameters

  New position in pixels.

- **bool sf::Window::pollEvent(Event & event)**

  Pop the event on top of the event queue, if any, and return it.

  This function is not blocking: if there's no pending event then it will return false and leave *event* unmodified. Note that more than one event may be present in the event queue, thus you should always call this function in a loop to make sure that you process every pending event.

  Parameters

  event| Event to be returned.

  Returns

  True if an event was returned, or false if the event queue was empty.

- **void sf::Window::setMouseCursorGrabbed(bool grabbed)**

  Grab or release the mouse cursor.

  If set, grabs the mouse cursor inside this window's client area so it may no longer be moved outside its bounds. Note that grabbing is only active while the window has focus.

  Parameters
  Grabbed| True to enable, false to disable.

- **void sf::Window::setSize(const Vector 2f & size)**

  Change the size of the rendering region of the window.

  Parameters

  Size| New size, in pixels.

- **void sf::Window::setTitle(const String & title)**

  Change the title of the window.

  Parameters

  Tilte| New title

```
while (window.pollEvent(event))
{
    if (event.type == Event::Closed)
        window.close();
    /*if (event.type == Event::MouseMoved)*/
    /*  cout << event.mouseMove.x << " " << event.mouseMove.y << endl;*/
    if (event.type == Event::MouseButtonPressed || event.type == Event::MouseButtonReleased)
    {
        movement(event, oldPosition, pos, newPosition, firstPlayer);
        if (RackCompleteCheck == 1)
        {
            P1.recordingMovement(oldPosition);
            if (P1.formingWord())
            {
                P1.Rack.completeRack1(Bag);
                //      P1.score();
                firstPlayer = 2;
                RackCompleteCheck = 0;
            }
        }
        if (RackCompleteCheck == 2)
        {
            P2.recordingMovement(oldPosition);
            if (P2.formingWord())
            {
                P2.Rack.completeRack2(Bag);
                //      P2.score();
                firstPlayer = 1;
                RackCompleteCheck = 0;
            }
```

3)Events:

Introduction:

- ### The sf::Event type:

sf::Event is a union, which means that only one of its members is valid at a. The valid member is the one that matches the event type, for example event.key for a KeyPressed event. Trying to read any other member will result in an undefined behavior. It is important to never try to use an event member that doesn't match its type.

sf::Event instances are filled by the pollEvent function of the sf::Window class. Only these two functions can produce valid events, any attempt to use an sf::Event which was not returned by successful call to pollEvent will result in the same undefined behavior.

```
while (window.pollEvent(event))
{
    if (event.type == Event::Closed)
        window.close();
    /*if (event.type == Event::MouseMoved)*/
    /*  cout << event.mouseMove.x << " " << event.mouseMove.y << endl;*/
    if (event.type == Event::MouseButtonPressed || event.type == Event::MouseButtonReleased)
    {
        movement(event, oldPosition, pos, newPosition, firstPlayer);
        if (RackCompleteCheck == 1)
        {
            P1.recordingMovement(oldPosition);
            if (P1.formingWord())
            {
                P1.Rack.completeRack1(Bag);
                //      P1.score();
                firstPlayer = 2;
                RackCompleteCheck = 0;
            }
        }
```

- **The Mouse Button Pressed and Released Events:**

  The sf::Event::MouseButtonPressed and sf::Event::MouseButtonReleased ev ents are triggered when a mouse button is pressed/released.

  SFML supports 5 mouse buttons: left, right, middle (wheel), extra #1 and extra #2 (side buttons).

  The member associated with these events is event.mouseButton, it contains the code of the pressed/released button, as well as the current position of the mouse cursor.

```
if (event.type == Event::MouseButtonPressed || event.type == Event::MouseButtonReleased)
```

- **The Closed Event:**

  The sf::Event::Closed event is triggered when the user wants to close the window, through any of the possible methods the window manager provides ("close" button, keyboard shortcut, etc.). This event only represents a close request, the window is not yet closed when the event is received.

  Typical code will just call window.close() in reaction to this event, to actually close the window. However, you may also want to do something else first, like saving the current application state or asking the user what to do. If you don't do anything, the window remains open.

  There's no member associated with this event in the sf::Event union.

```
if (event.type == Event::Closed)
    window.close();
```

4)Keyboard and Mouse:

Introduction:

Below is explained how to access global input devices keyboard and mouse in SFML.

- **Keyboard:**

  The class that provides access to the keyboard state is sf::Keyboard. It only contains one function, isKeyPressed, which checks the current state of a key (pressed or released). It is a static function, so you don't need to instantiate sf::Keyboard to use it.This function directly reads the keyboard state, ignoring the focus state of your window. This means that isKeyPressed may return true even if your window is inactive.

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
{
    // left key is pressed: move our character
    character.move(1, 0);
}
```

  Key codes are defined in the sf::Keyboard::Key enum.

- **Mouse:**

  The class that provides access to the mouse state is sf::Mouse. Like its friend sf::Keyboard, sf::Mouse only contains static functions and is not meant to be instantiated (SFML only handles a single mouse for the time being).Mouse button codes are defined in the sf::Mouse::Button enum. SFML supports up to 5 buttons: left, right, middle (wheel), and two additional buttons whatever they may be.

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
    // left mouse button is pressed: shoot
    gun.fire();
}
```

Detailed Explanation of Graphics Module of Multi-Media In SFML

1)Drawing 2D Stuff:

Introduction:

SFML provides a graphics module which will help you draw 2D entities in a much simpler way than with OpenGL.

- **The Drawing Window:**

  To draw the entities provided by the graphics module, you must use a specialized window class: sf::RenderWindow. This class is derived from sf::Window, and inherits all its functions. sf::RenderWindow adds high-level functions to help you draw things easily.We will focus on two functions: clear and draw. They are as simple as their name implies: clear clears the whole window with the chosen color, and draw draws whatever object you pass to it. Calling clear before drawing anything is mandatory, otherwise the contents from previous frames will be present behind anything you draw. Calling display is also mandatory, it takes what was drawn since the last call to display and displays it on the window. Indeed, things are not drawn directly to the window, but to a hidden buffer. This buffer is then copied to the window when you call display-- this is called double-buffering.

- **What can be drawn Now?**

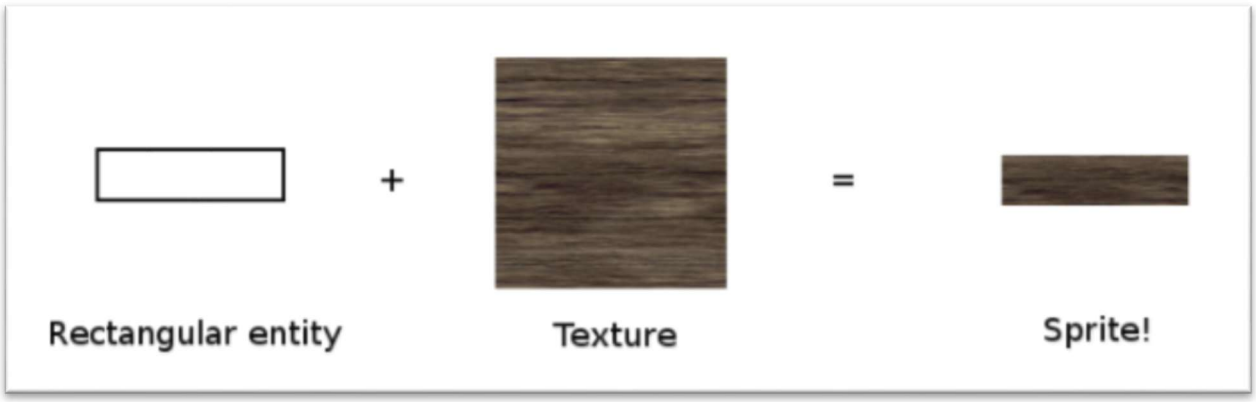
  SFML provides following drawable entities:
  - Sprites
  - Text
  - Shapes

```cpp
#include <SFML/Graphics.hpp>

int main()
{
    // create the window
    sf::RenderWindow window(sf::VideoMode(800, 600), "My window");

    // run the program as long as the window is open
    while (window.isOpen())
    {
        // check all the window's events that were triggered since the last iteration of the loop
        sf::Event event;
        while (window.pollEvent(event))
        {
            // "close requested" event: we close the window
            if (event.type == sf::Event::Closed)
                window.close();
        }

        // clear the window with black color
        window.clear(sf::Color::Black);

        // draw everything here...
        // window.draw(...);

        // end the current frame
        window.display();
    }

    return 0;
}
```

2)Sprites and Textures:

- **Sprites:**

  Most (if not all) of you are already familiar with these two very common objects, so let's define them very briefly. A texture is an image. But we call it "texture" because it has a very specific role: being mapped to a 2D entity. A sprite is nothing more than a textured rectangle.



- **Textures:**

  Before creating any sprite, we need a valid texture. The class that encapsulates textures in SFML is, surprisingly, sf::Texture. Since the only role of a texture is to be loaded and mapped to graphical entities, almost all its functions are about loading and updating it. The most common way of loading a texture is from an image file on disk, which is done with the loadFromFile function.

```
f (!keys[1].loadFromFile("0.jpg"))
    cout << "0 error";
or (int i = 2; i < 11; i++)
f (!keys[i].loadFromFile("A.jpg"))
    cout << "A error";
f (!keys[11].loadFromFile("B.jpg"))
    cout << "B error";
f (!keys[12].loadFromFile("B.jpg"))
    cout << "B error";
```

```
table.loadFromFile("Board.jpg");
tableLayout.setTexture(table);
background.loadFromFile("Background.jpg");
backgroundlayout.setTexture(background);
DoneButton.loadFromFile("Done.jpg");
DoneButtonLayout1.setTexture(DoneButton);
DoneButtonLayout2.setTexture(DoneButton);
```

```
for (int i = 0; i < 100; i++)
{
    keysLayout[i].setTexture(keys[i], true);
}
```

3)Texts and Fonts:

- ## Loading Fonts:

  Before drawing any text, you need to have an available font, just like any other program that prints text. Fonts are encapsulated in the sf::Font class. The most common way of loading a font is from a file on disk, which is done with the loadFromFile function. If you want to load a font, you will need to include the font file with your application, just like every other resource.

- ## Drawing Texts:

  To draw text, you will be using the sf::Text class.

```
sf::Text text;

// select the font
text.setFont(font); // font is a sf::Font

// set the string to display
text.setString("Hello world");

// set the character size
text.setCharacterSize(24); // in pixels, not points!

// set the color
text.setFillColor(sf::Color::Red);

// set the text style
text.setStyle(sf::Text::Bold | sf::Text::Underlined);

...

// inside the main loop, between window.clear() and window.display()
window.draw(text);
```

4)Shapes:

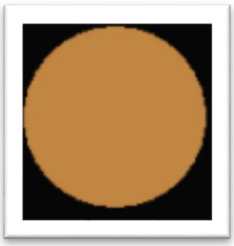
Introduction:

SFML provides a set of classes that represent simple shape entities. Each type of shape is a separate class, but they all derive from the same base class so that they have access to the same subset of common features. Each class then adds its own specifics: a radius property for the circle class, a size for the rectangle class, points for the polygon class, etc.

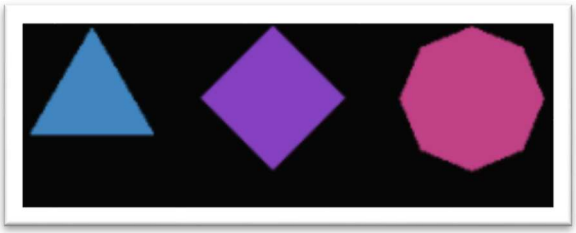
- **Common Shape Properties:**
  - Color
  - Outline
  - Texture

- **Built-in Shape Types:**
  - Rectangles

  - Circles



  - Rectangular Polygons



- Convex Shapes

- Lines

5)Transforming Entities:

Introduction:

sf::Transformable (and all its derived classes) defines four properties: **position**, **rotation**, **scale** and **origin**. They all have their respective getters and setters. These transformation components are all independent of one another: If you want to change the orientation of the entity, you just have to set its rotation property, you don't have to care about the current position and scale.

- **Position:**
  The position is the position of the entity in the 2D world.

```
PlayerTile[i]->setPosition(0, 39 + (i * 40));
keysLayout[x[i]].setPosition(0, 39 + (i * 40));
```

Dependencies of SFML:

SFML depends on a few other libraries, so before starting to compile you must have their development files installed. On Windows all the needed dependencies are provided directly with SFML, so you don't have to download/install anything. SFML has also internal dependencies: Audio and Window depend on System, while Graphics depends on System and Window. In order to use the Graphics module, you must link with Graphics, Window, and System (the order of linkage matters with GCC).

Basics required to program with SFML:

- Compiling, building
- Basic program structure (main(), header, pre-processor directives etc)
- Basic data types
- Composite data types
- Control structures (if, for, while etc)
- Basic functions
- Function parameter passing
- Classes and general Object-Oriented Programming
- STL - Standard Template Library
- Dynamic memory allocation, pointers
- Type casting
- **Debugging techniques** This is important to be able to help yourself when the situation arises.
- Templates
- Operator overloading
- Namespaces
- Move semantics

Data Structures and Algorithms:

The data structures and algorithms that we have used during the process of making our Computerized Two Player Scrabble Game are Arrays(basic) and most importantly tries.

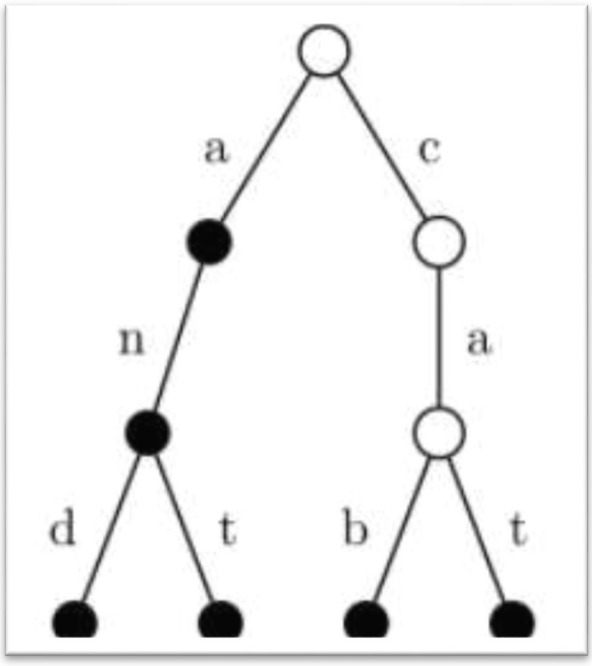
Arrays:

Array is a data structure which stores a fixed-size sequential collection of elements/variables of the same type. C++ allows multidimensional arrays i.e. 2D and 3D arrays.

```
char tableValues[15][15];
char tempTableValues[15][15];
```

Tries:

A trie is a tree whose edges are labeled with letters. Paths from the root of the trie spell out words. The image below shows a trie.



The black nodes indicate that the path to this point in the trie spells a word. The words represented by this trie are *a*, *an*, *and*, *ant*, *cab* and *cat*. It is very fast to check whether a word is in a trie: one need only start from the root and trace out the word along the edges; if one ends up at a word-terminating node (a black node), then the word is in the trie, otherwise it is not. A trie is very good for spell checking: simply construct a trie representing all the words in a dictionary, and then check each word in a document to see if it is in the trie. The words not in the trie are (presumably) misspelled.
Tries can also efficiently support auto-completion. Given the prefix-based structure of the trie, it is straightforward to enumerate all of the words that begin with a given letter sequence: these will be represented by nodes that are descendants of the node corresponding to that prefix.

```
struct TrieNode
{
    TrieNode * child[NO_OF_ALPHABETS];
    bool is_End_of_Word;
};

class trie
{
private:
    TrieNode* rooot;
public:
    trie()
    {
        rooot = newNode();
    }

    TrieNode* newNode()
    {
        TrieNode* temp = new TrieNode;
        for (int i = 0; i < NO_OF_ALPHABETS; i++)
            temp->child[i] = nullptr;
        temp->is_End_of_Word = false;
        return temp;

    }
```

Outline Solution:

Our program consists of one cpp file. The classes, constructors, destructors and functions are outlined below:

- **Main();** to create objects and to call functions. It serves the purpose of calling all functions, inputting all libraries, calling clear screen and system("pause") functions and function for accessing the system time.

- **PremiumWords();**

- **struct TrieNode{};**

- **class trie{};**
  - trie(){}
  - newNode();
  - insert(string);
  - bool search(string);

- **void alphabets();**

- **void loadImages();**

- **void tempArray();**

- **void movement();**

- **class graphicTile{};**
  - graphicTile(){}
  - setX(int);
  - setY(int);
  - getY();

- **class Tile{};**
  - Tile(){}
  - Tile(int,char,int,int,int);
  - setPosition(int,int);
  - getLetter();
  - getValue();

- **class BagOfTiles{};**
  - BagOfTiles (){}
  - getNoOfTiles();
  - getTile(int,int);
  - showAllBag();
  - ~BagOfTiles();

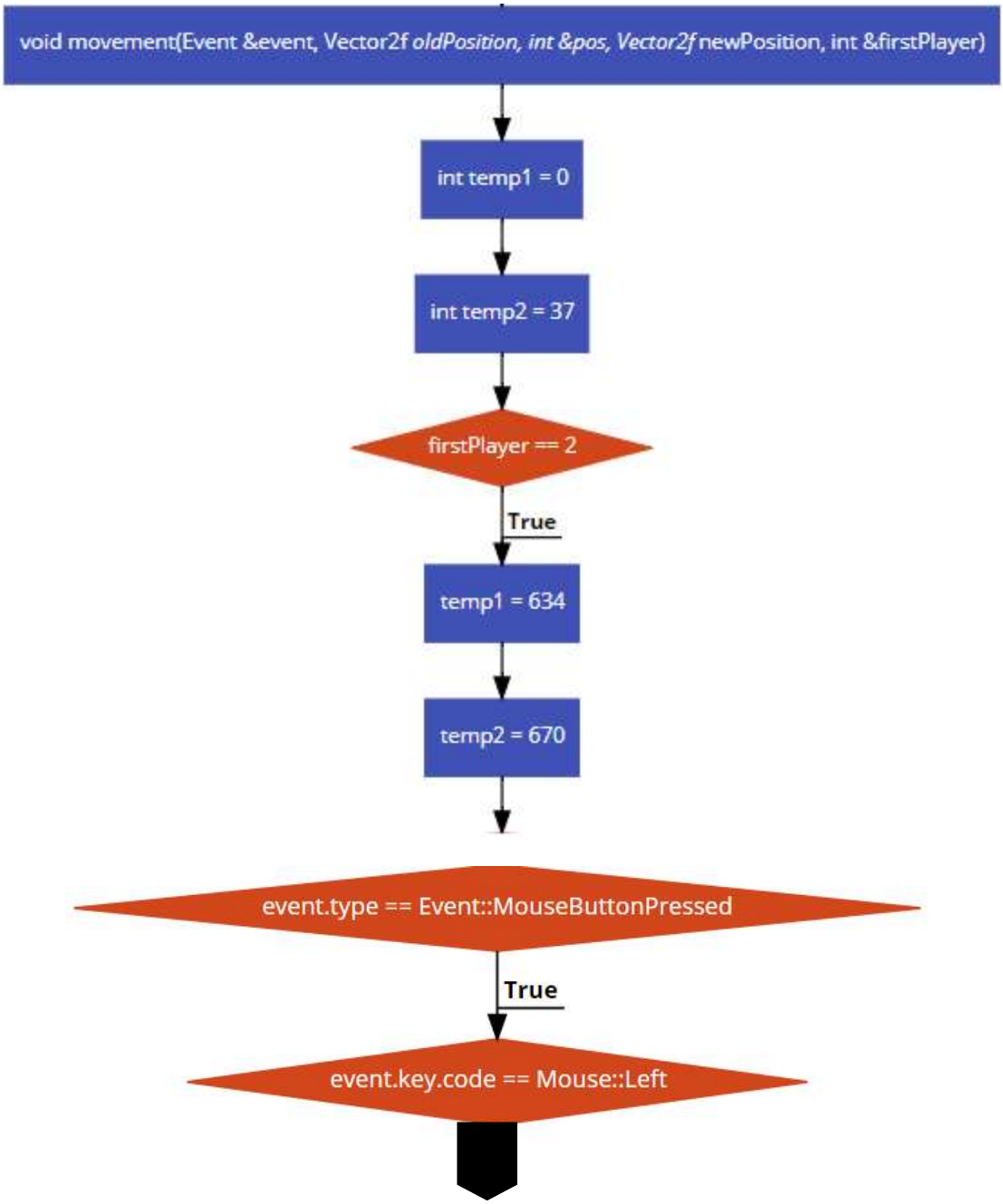
- **class RackOfTiles{};**
  - RackOfTiles(){}
  - completeRack1();
  - completeRack2();
  - moveRack1();
  - moveRack2();
  - showAllRack();
  - ~RackOfTiles();

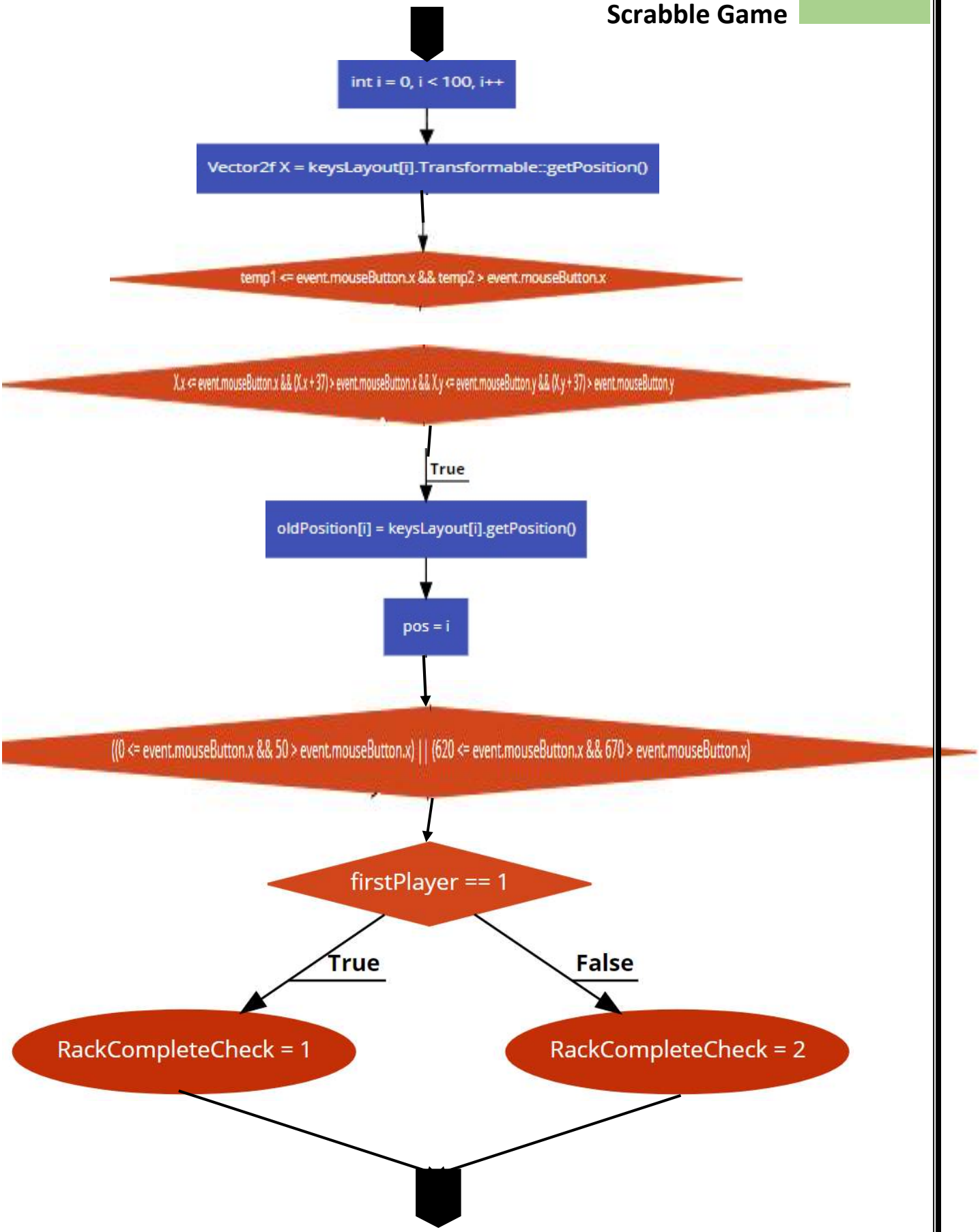
- **class Player{};**
  - Player(){}
  - IntPlayerRack(BagOfTiles &,int);
  - RecordingMovement(Vector2f*);
  - formingWord();
  - validatingWord(string);
  - score();

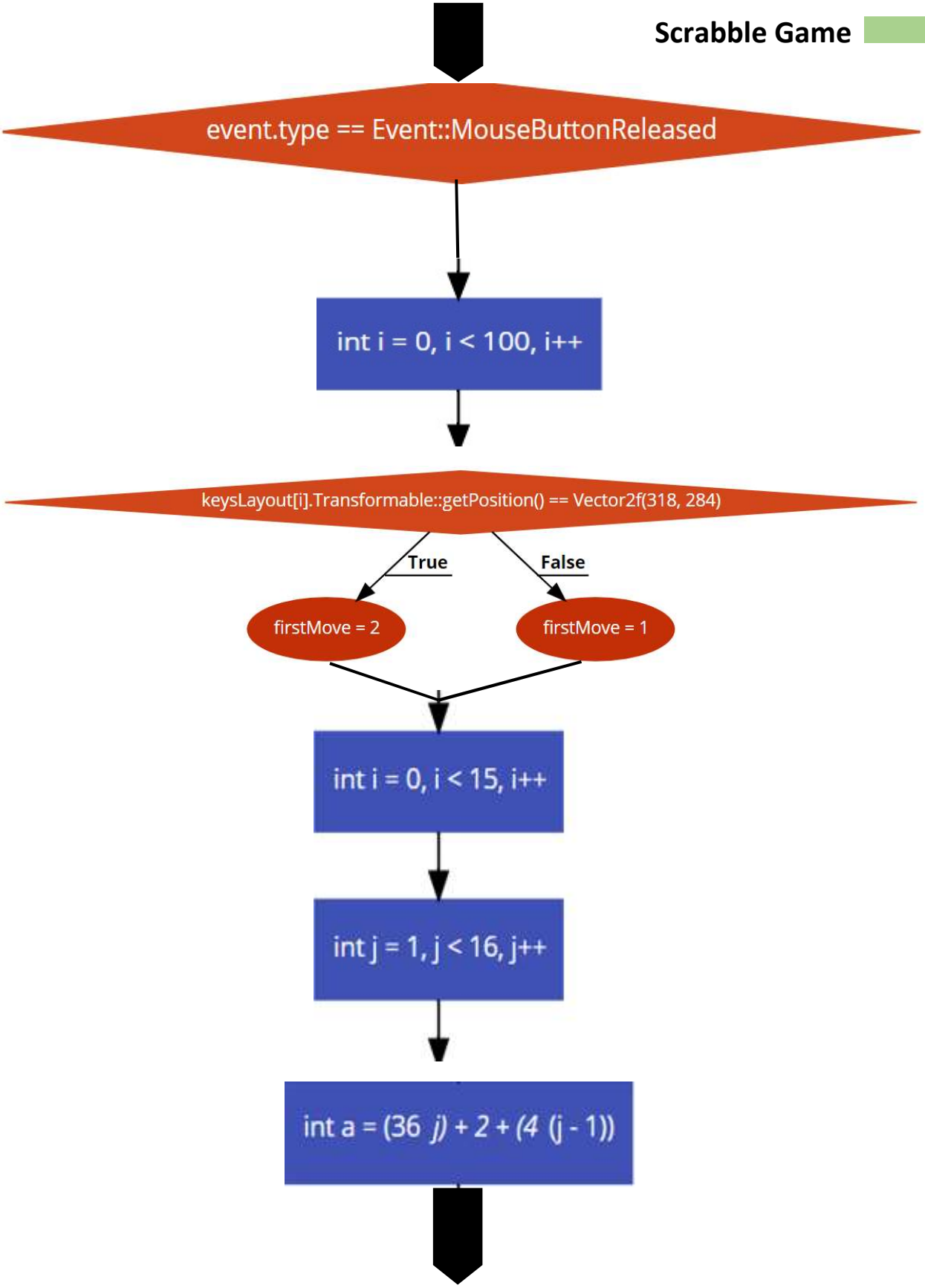
Flowcharts

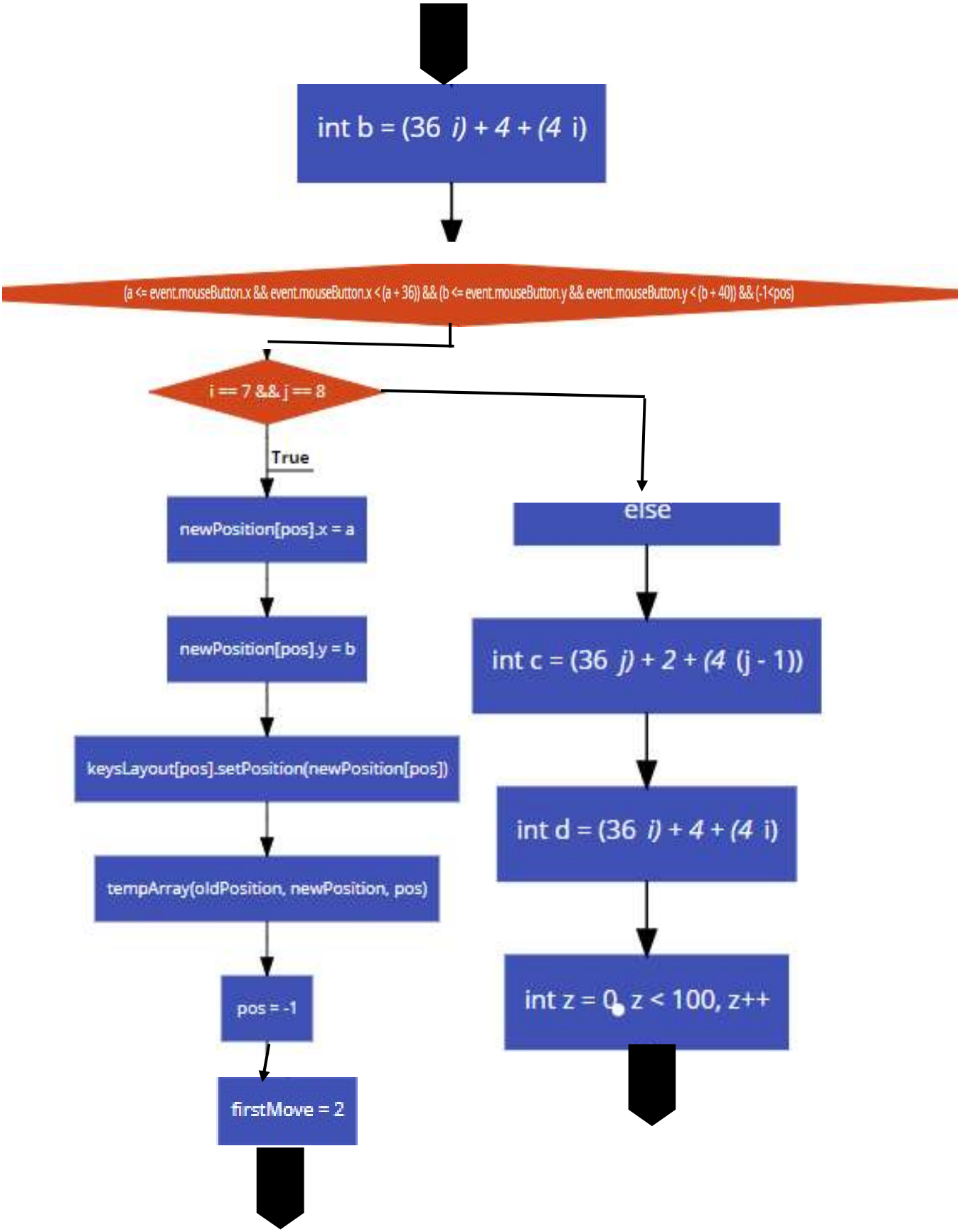
Flowchart showing basic working of Scrabble Game:

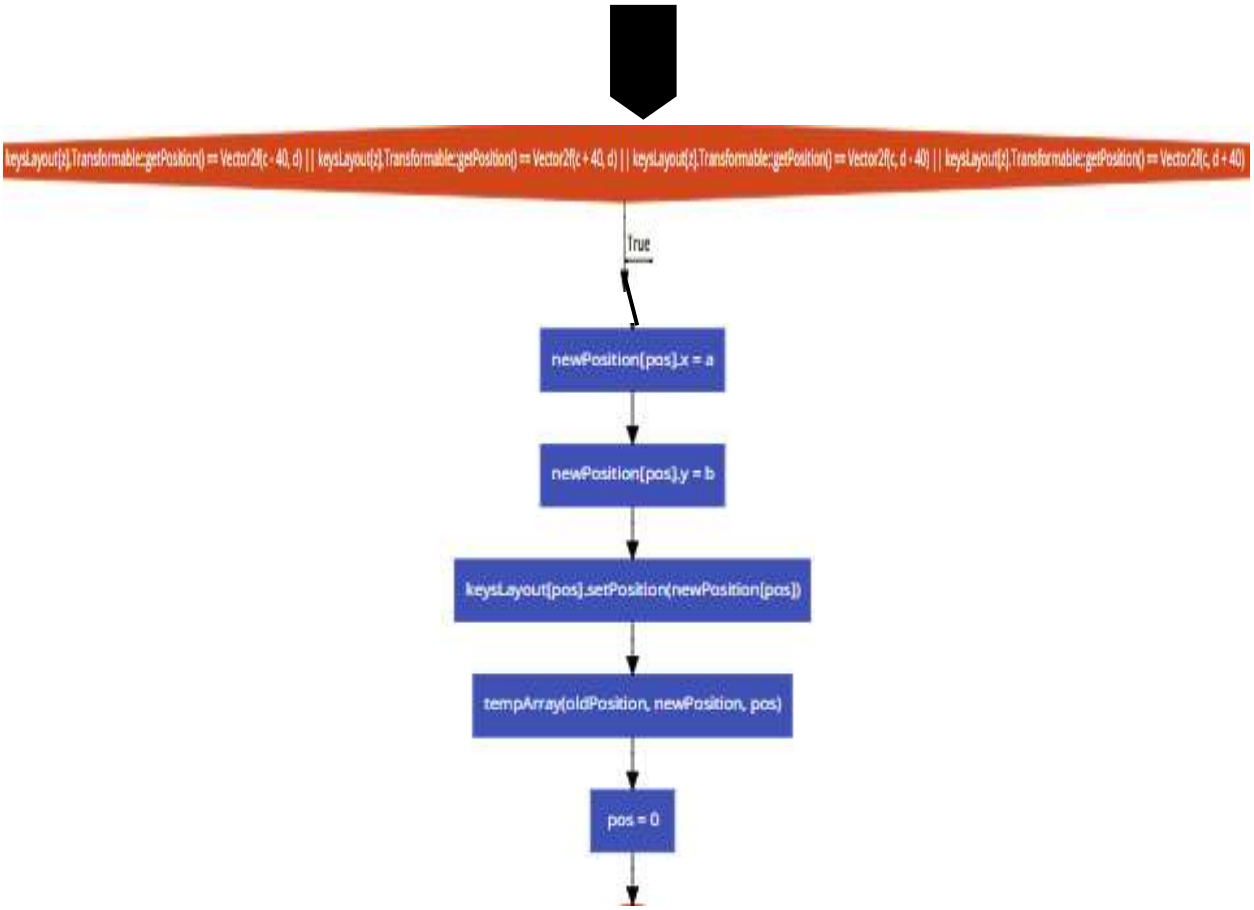
Movement();



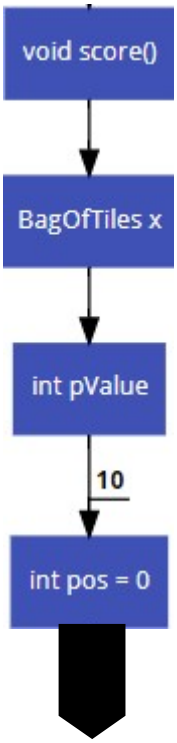

int i = 0, i < 100, i++
Vector2f X = keysLayout[i].Transformable::getPosition()
temp1 <= event.mouseButton.x && temp2 > event.mouseButton.x
X.x <= event.mouseButton.x && (X.x + 37) > event.mouseButton.x && X.y <= event.mouseButton.y && (X.y + 37) > event.mouseButton.y
True
oldPosition[i] = keysLayout[i].getPosition()
pos = i
((0 <= event.mouseButton.x && 50 > event.mouseButton.x) || (620 <= event.mouseButton.x && 670 > event.mouseButton.x)
firstPlayer == 1
True
False
RackCompleteCheck = 1
RackCompleteCheck = 2

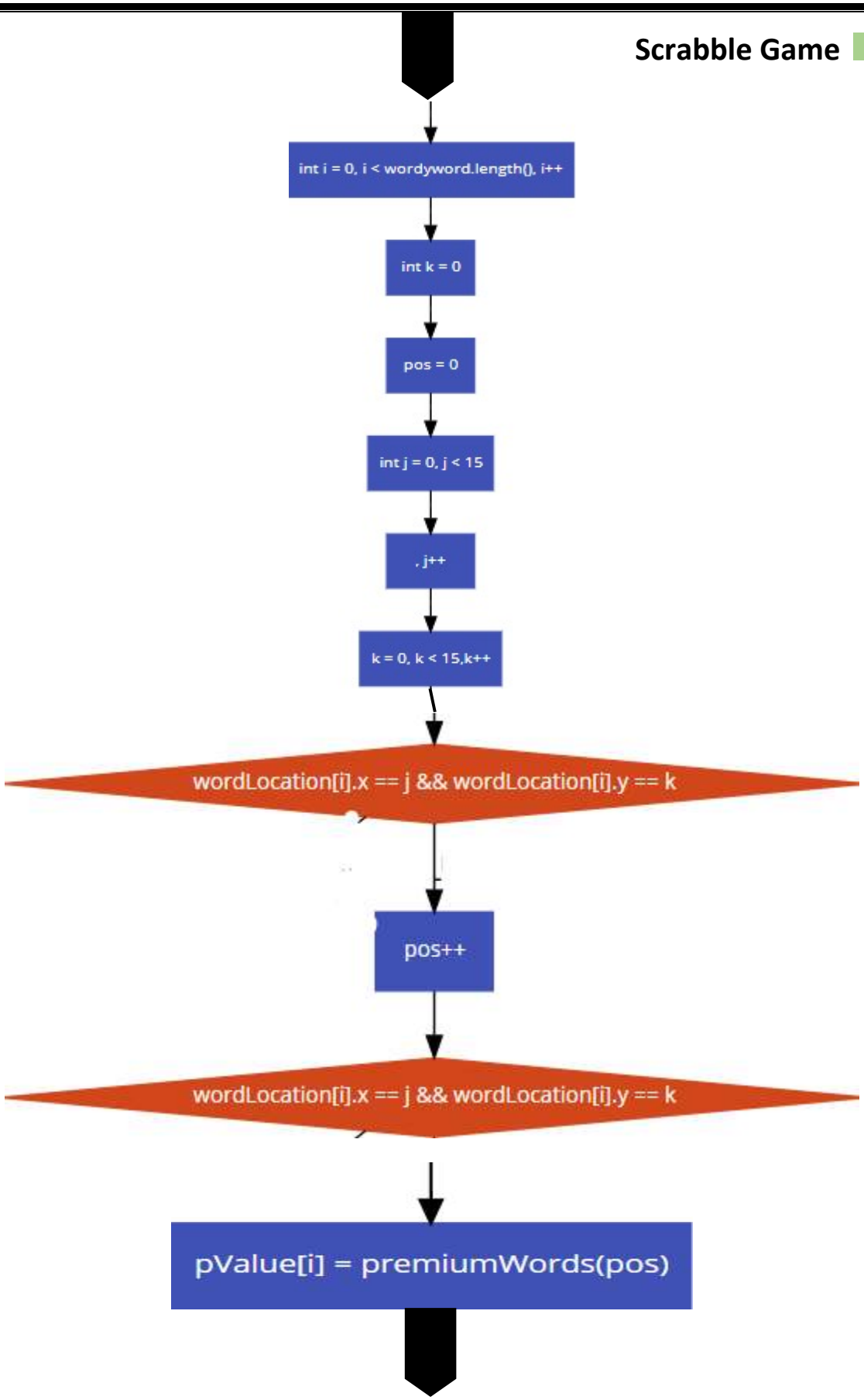

event.type == Event::MouseButtonReleased
int i = 0, i < 100, i++
keysLayout[i].Transformable::getPosition() == Vector2f(318, 284)
True
False
firstMove = 2
firstMove = 1
int i = 0, i < 15, i++
int j = 1, j < 16, j++
int a = (36 j) + 2 + (4 (j - 1))

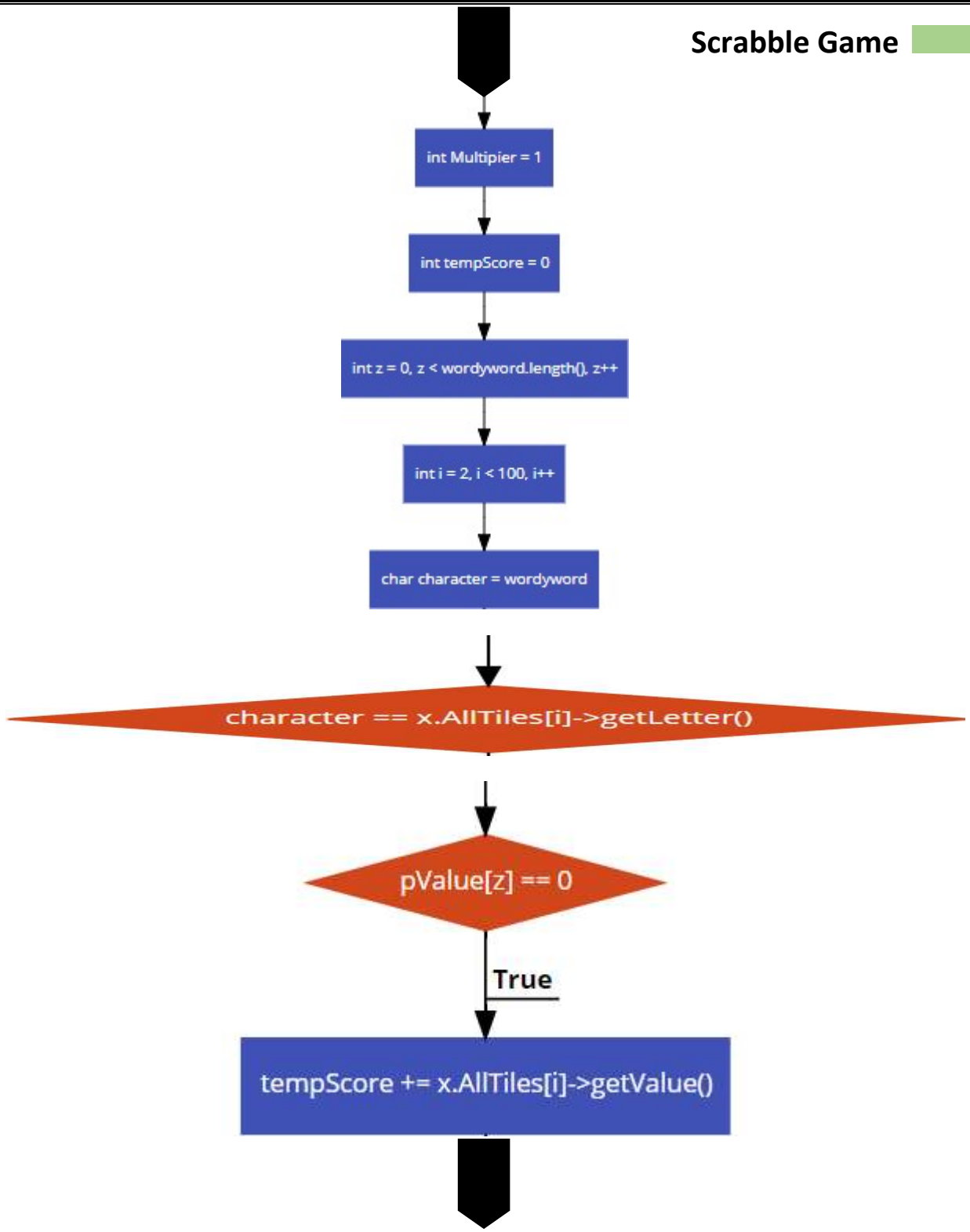

int b = (36 i) + 4 + (4 i)
(a <= event.mouseButton.x && event.mouseButton.x < (a + 36)) && (b <= event.mouseButton.y && event.mouseButton.y < (b + 40)) && (-1<pos)
i == 7 && j == 8
True
newPosition[pos].x = a
newPosition[pos].y = b
keysLayout[pos].setPosition(newPosition[pos])
tempArray(oldPosition, newPosition, pos)
pos = -1
firstMove = 2
else
int c = (36 j) + 2 + (4 (j - 1))
int d = (36 i) + 4 + (4 i)
int z = 0, z < 100, z++

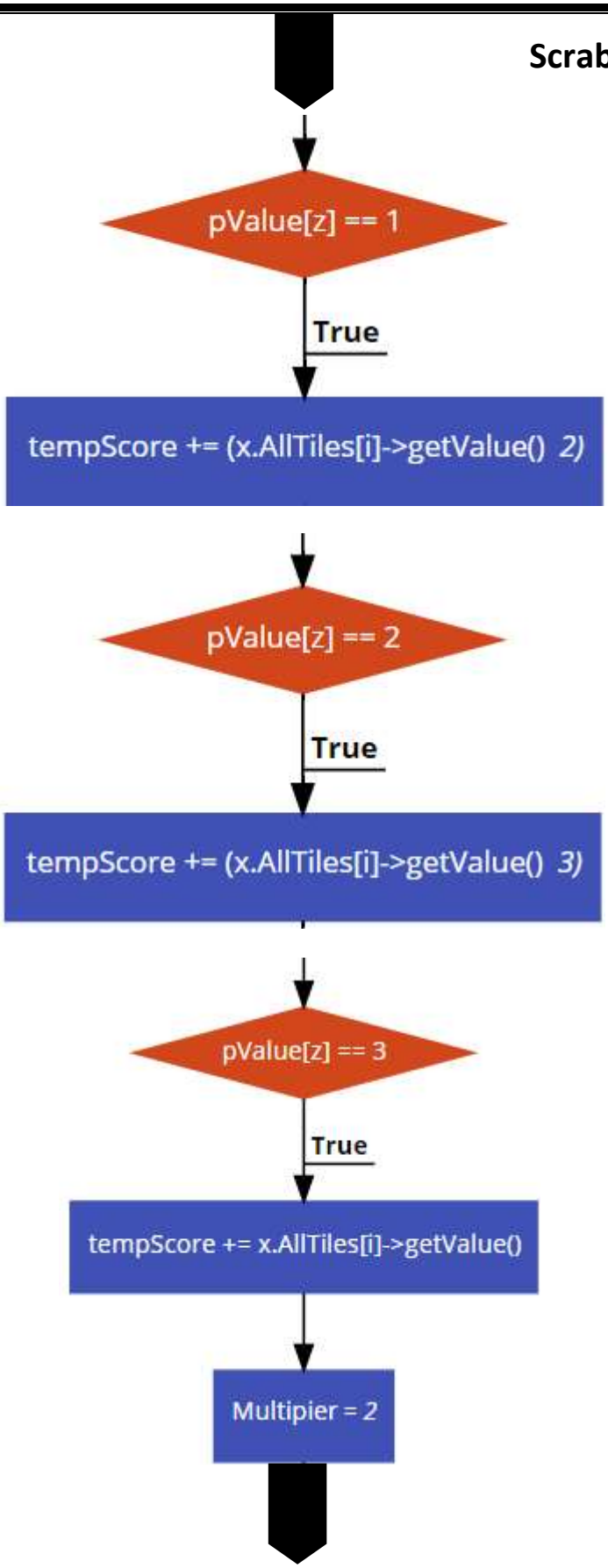

keysLayout[z].Transformable::getPosition() == Vector2f(c - 40, d) || keysLayout[z].Transformable::getPosition() == Vector2f(c + 40, d) || keysLayout[z].Transformable::getPosition() == Vector2f(c, d - 40) || keysLayout[z].Transformable::getPosition() == Vector2f(c, d + 40)
True
newPosition[pos].x = a
newPosition[pos].y = b
keysLayout[pos].setPosition(newPosition[pos])
tempArray(oldPosition, newPosition, pos)
pos = 0

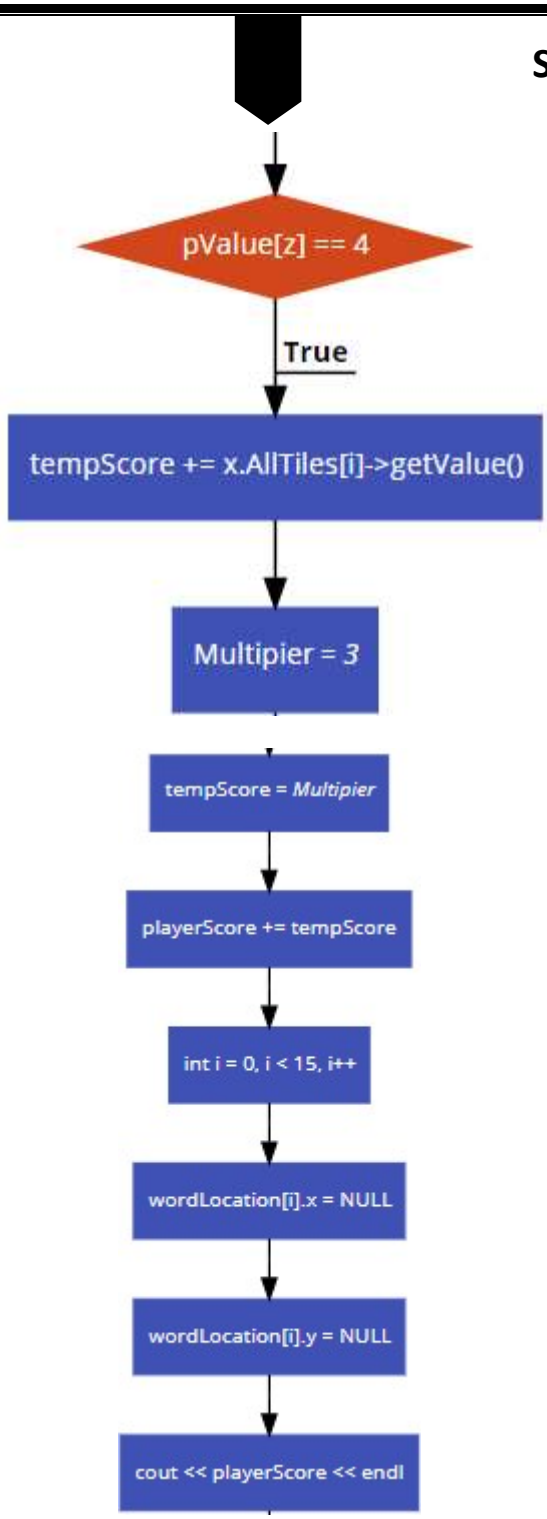
Score();


void score()
BagOfTiles x
int pValue
10
int pos = 0

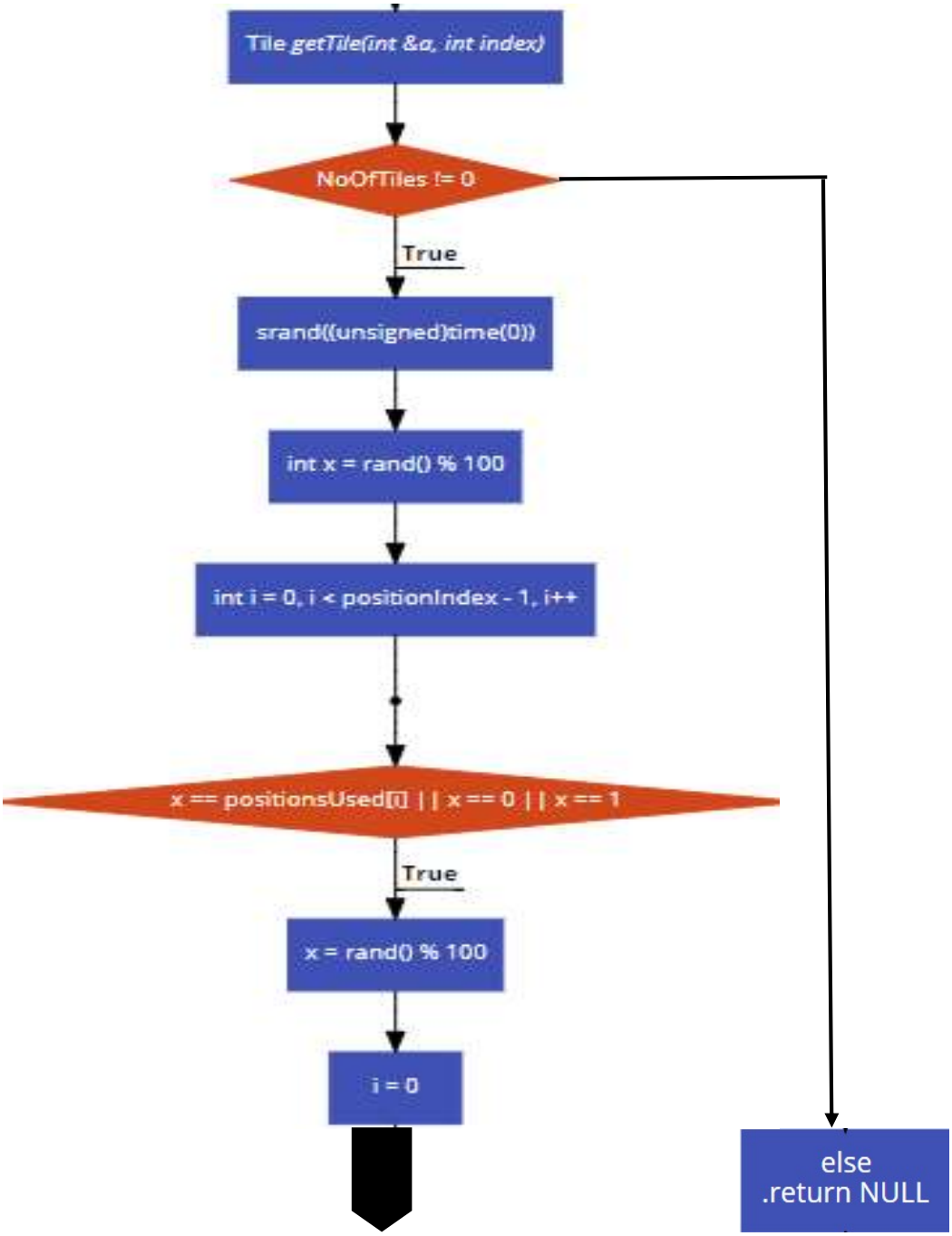

int i = 0, i < wordyword.length(), i++
int k = 0
pos = 0
int j = 0, j < 15
, j++
k = 0, k < 15,k++
wordLocation[i].x == j && wordLocation[i].y == k
pos++
wordLocation[i].x == j && wordLocation[i].y == k
pValue[i] = premiumWords(pos)

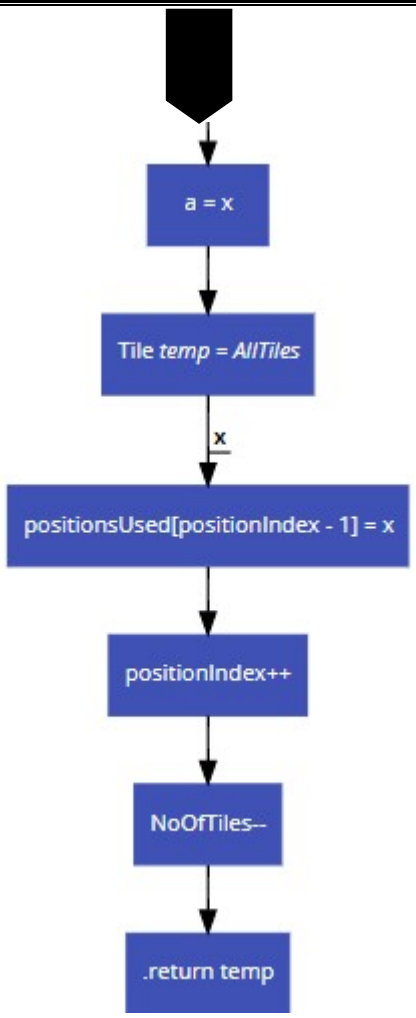

int Multipier = 1
int tempScore = 0
int z = 0, z < wordyword.length(), z++
int i = 2, i < 100, i++
char character = wordyword
character == x.AllTiles[i]->getLetter()
pValue[z] == 0
True
tempScore += x.AllTiles[i]->getValue()


pValue[z] == 1
True
tempScore += (x.AllTiles[i]->getValue() 2)
pValue[z] == 2
True
tempScore += (x.AllTiles[i]->getValue() 3)
pValue[z] == 3
True
tempScore += x.AllTiles[i]->getValue()
Multipier = 2

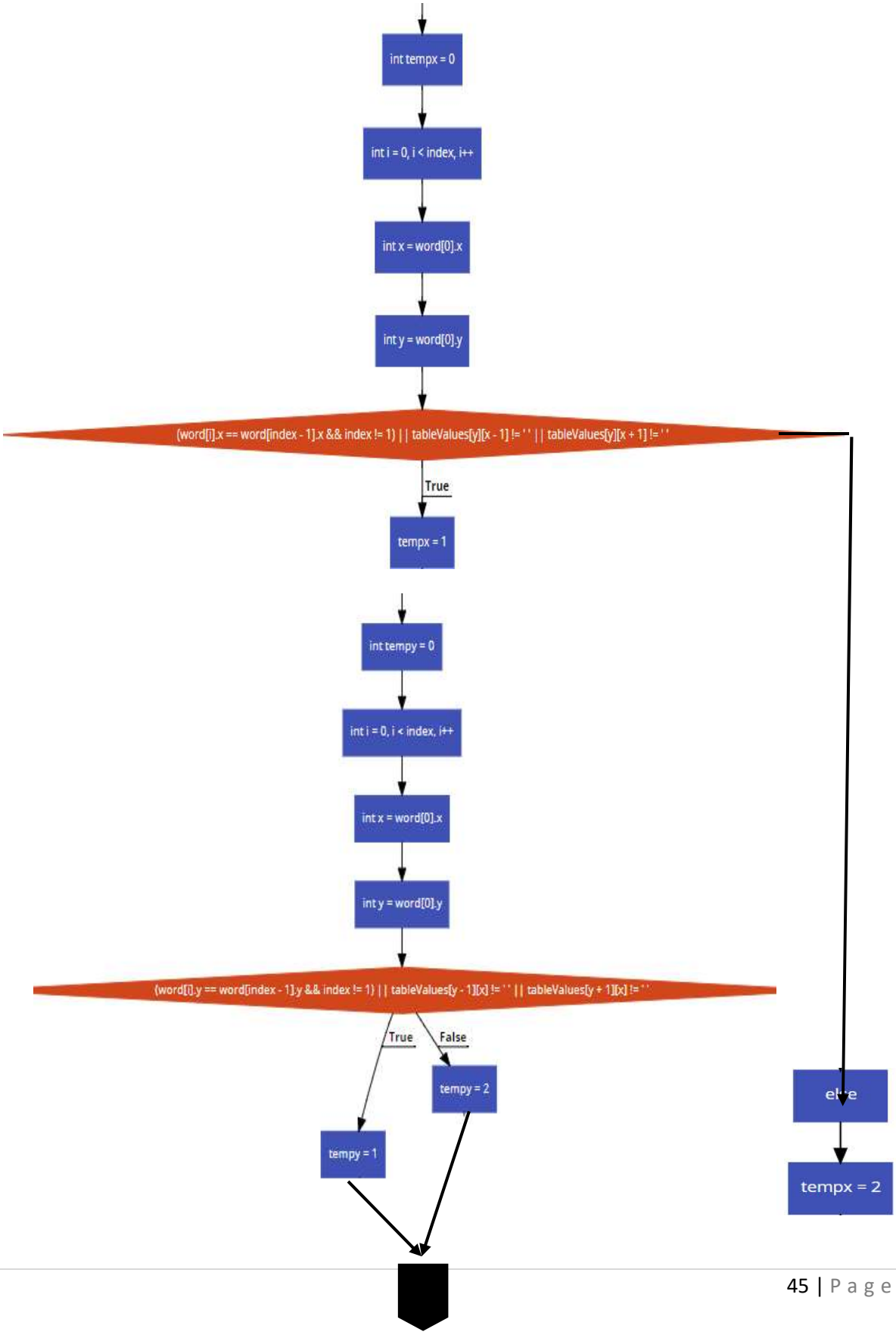

pValue[z] == 4
True
tempScore += x.AllTiles[i]->getValue()
Multipier = 3
tempScore = Multipier
playerScore += tempScore
int i = 0, i < 15, i++
wordLocation[i].x = NULL
wordLocation[i].y = NULL
cout << playerScore << endl

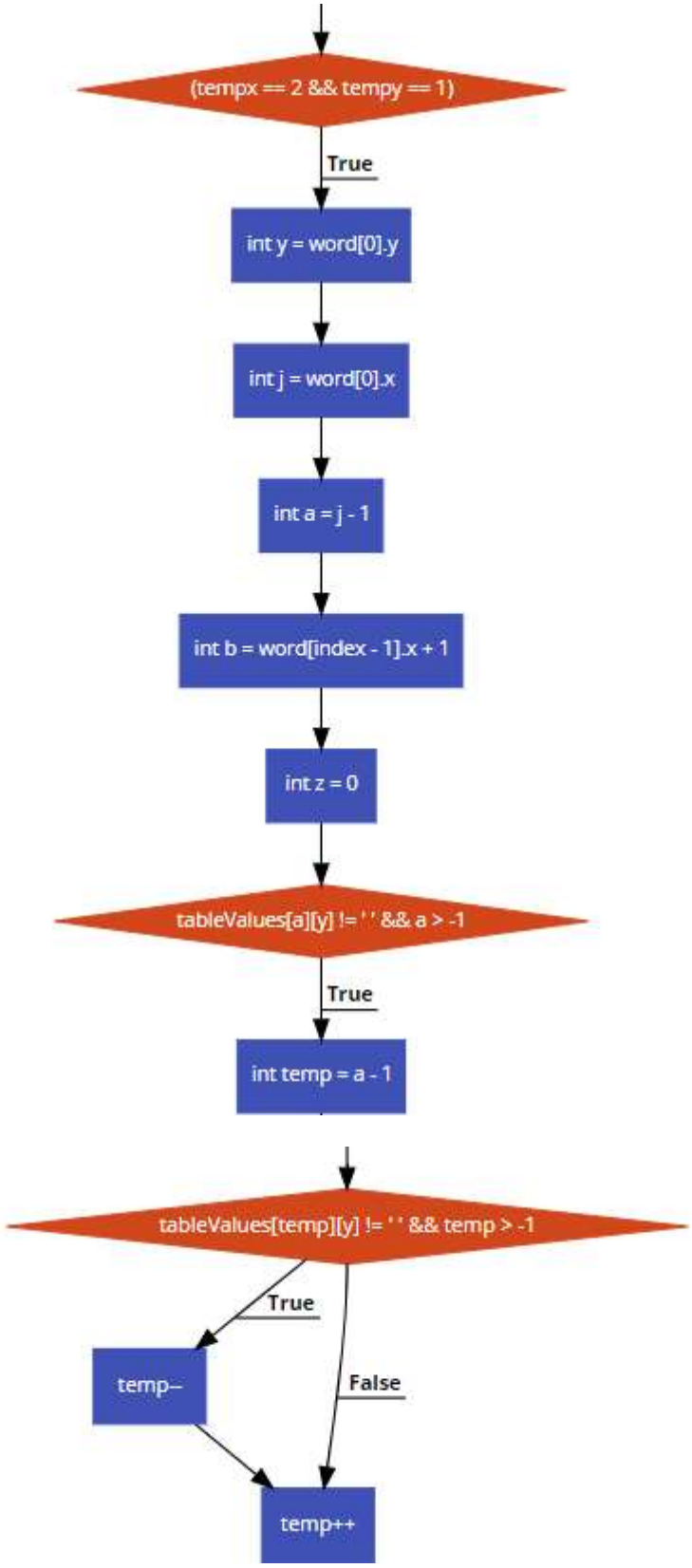
Tile*getTile(int,int);

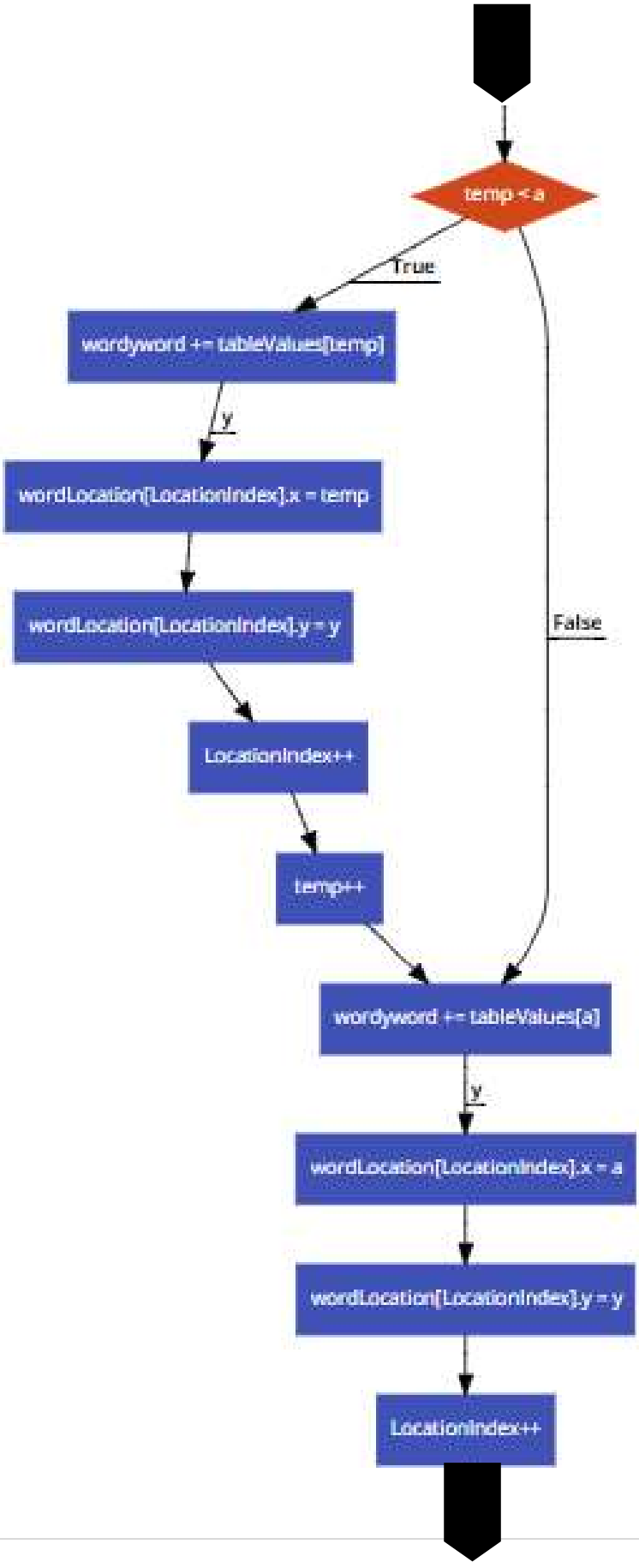


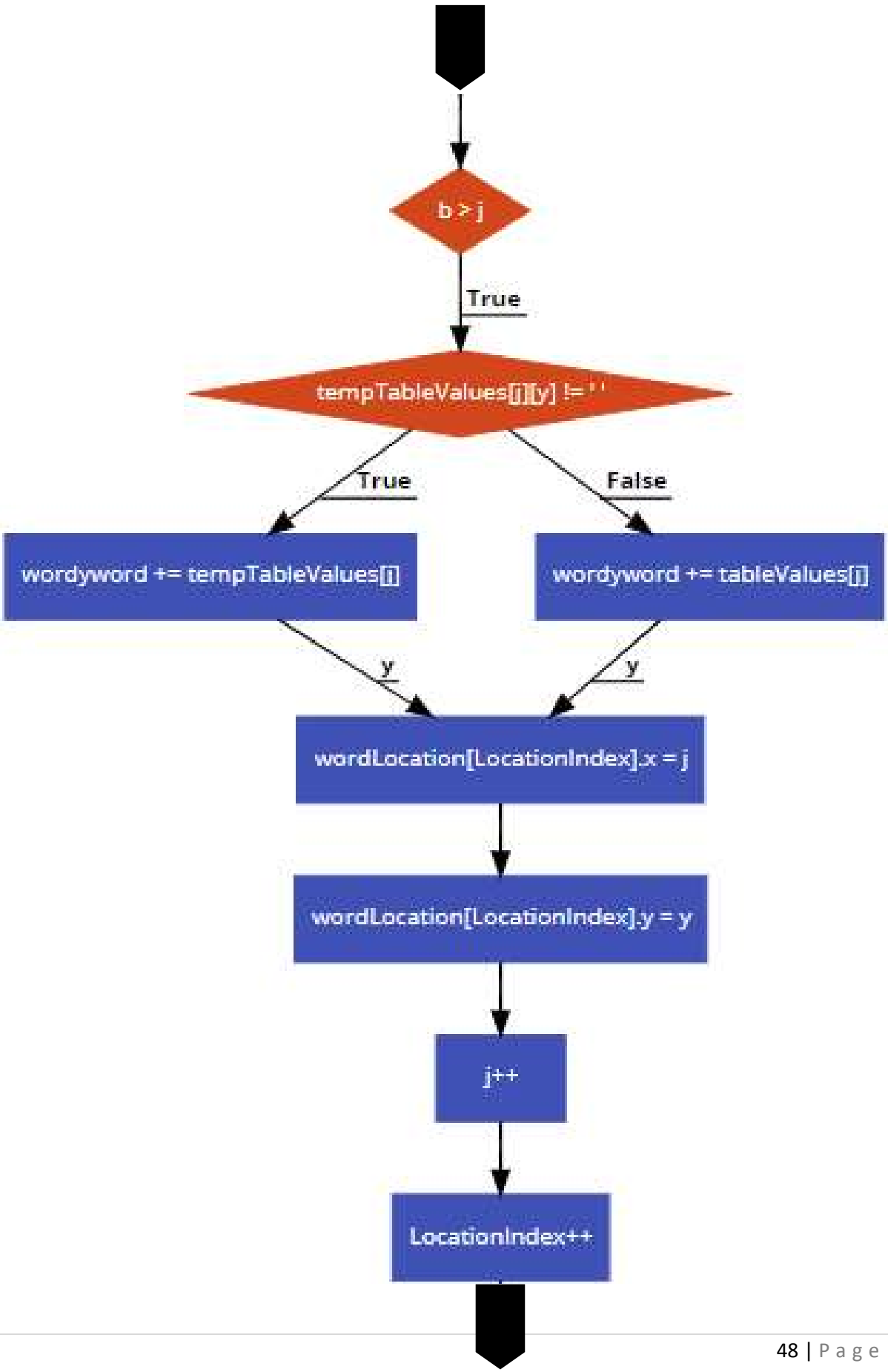


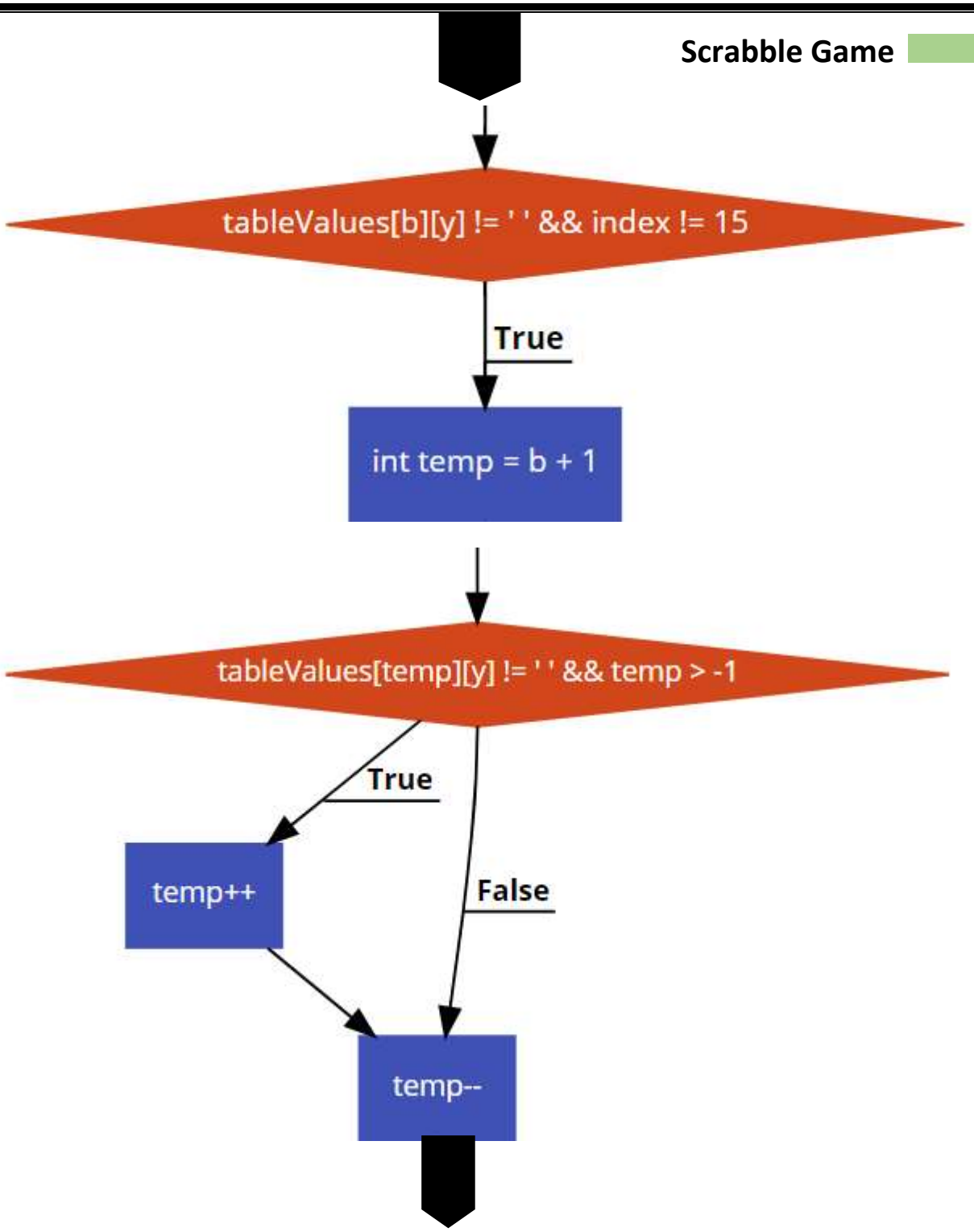
Bool formingWord()

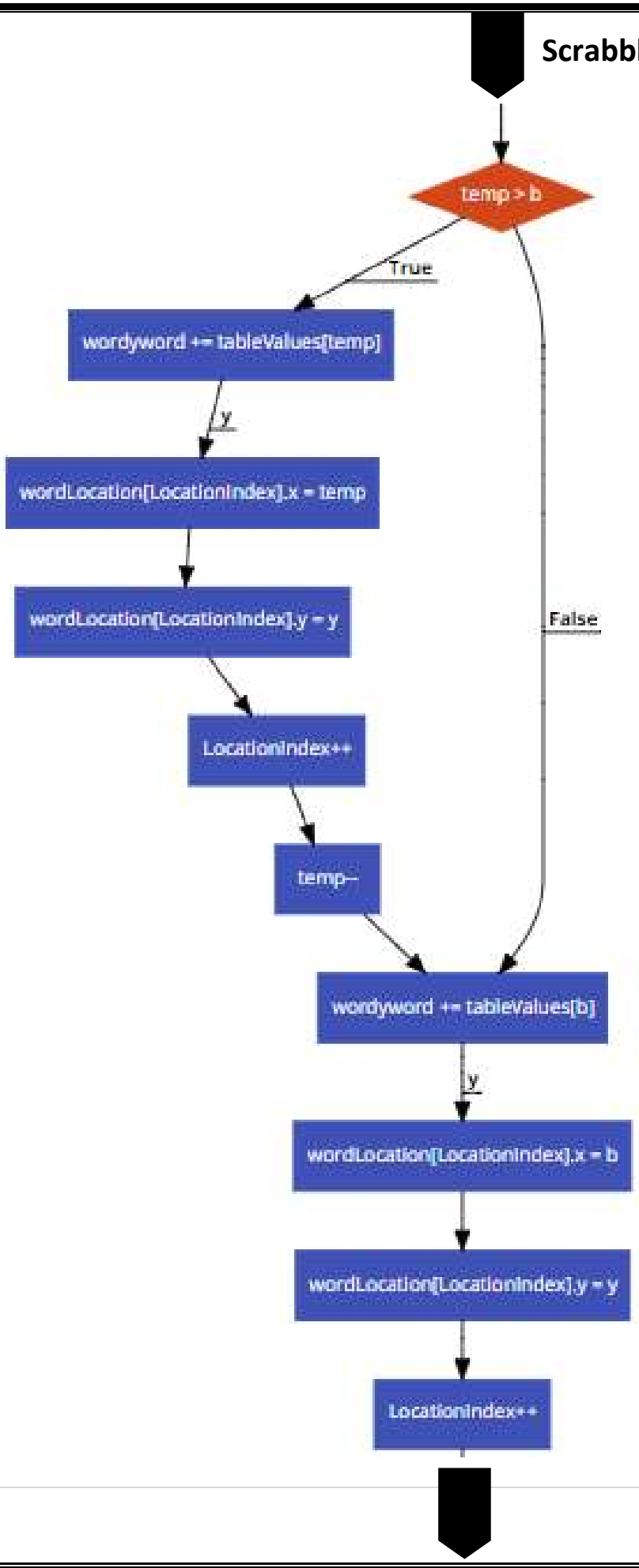

int tempx = 0
int i = 0, i < index, i++
int x = word[0].x
int y = word[0].y
(word[i].x == word[index - 1].x && index != 1) || tableValues[y][x - 1] != ' ' || tableValues[y][x + 1] != ' '
True
tempx = 1
int tempy = 0
int i = 0, i < index, i++
int x = word[0].x
int y = word[0].y
(word[i].y == word[index - 1].y && index != 1) || tableValues[y - 1][x] != ' ' || tableValues[y + 1][x] != ' '
True
False
tempy = 2
tempy = 1
else
tempx = 2

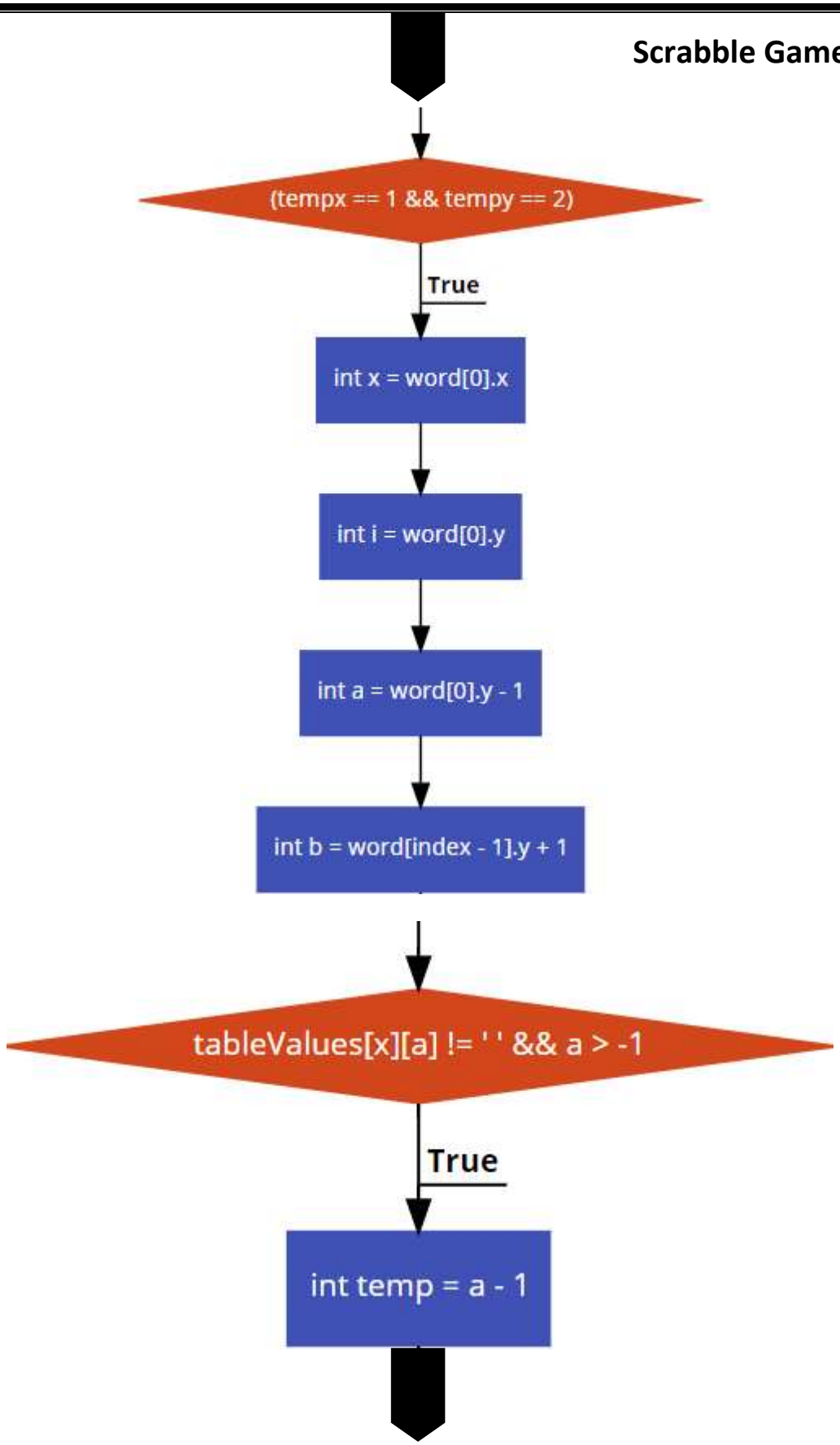

(tempx == 2 && tempy == 1)
True
int y = word[0].y
int j = word[0].x
int a = j - 1
int b = word[index - 1].x + 1
int z = 0
tableValues[a][y] != ' ' && a > -1
True
int temp = a - 1
tableValues[temp][y] != ' ' && temp > -1
True
False
temp--
temp++

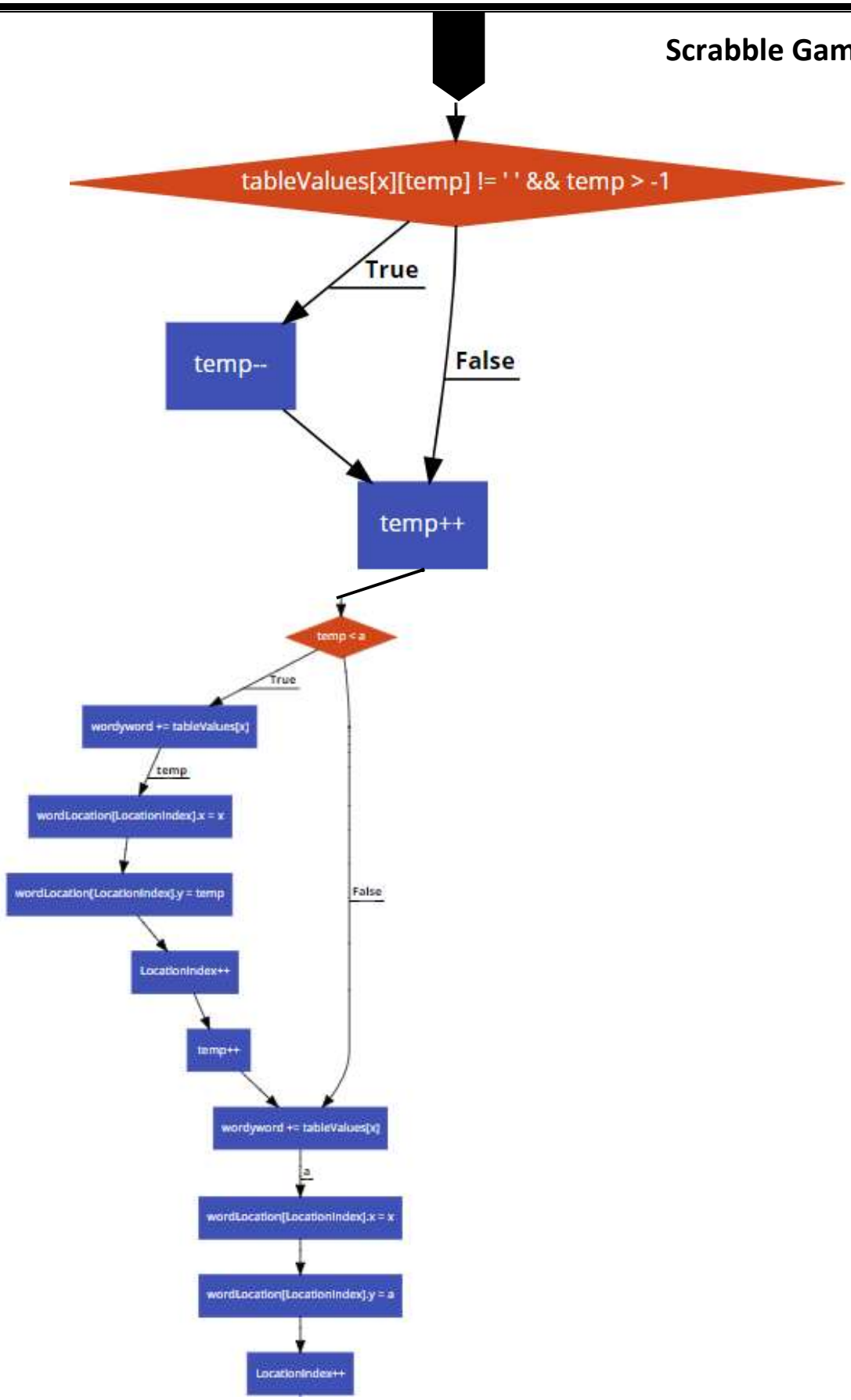

temp < a
True
wordyword += tableValues[temp]
y
wordLocation[LocationIndex].x = temp
wordLocation[LocationIndex].y = y
LocationIndex++
temp++
False
wordyword += tableValues[a]
y
wordLocation[LocationIndex].x = a
wordLocation[LocationIndex].y = y
LocationIndex++


b > j
True
tempTableValues[j][y] != ' '
True
False
wordyword += tempTableValues[j]
wordyword += tableValues[j]
y
y
wordLocation[LocationIndex].x = j
wordLocation[LocationIndex].y = y
j++
LocationIndex++


tableValues[b][y] != ' ' && index != 15
True
int temp = b + 1
tableValues[temp][y] != ' ' && temp > -1
True
False
temp++
temp--


temp > b
True
wordyword += tableValues[temp]
y
wordLocation[LocationIndex].x = temp
wordLocation[LocationIndex].y = y
LocationIndex++
temp--
False
wordyword += tablevalues[b]
y
wordLocation[LocationIndex].x = b
wordLocation[LocationIndex].y = y
LocationIndex++


(tempx == 1 && tempy == 2)
True
int x = word[0].x
int i = word[0].y
int a = word[0].y - 1
int b = word[index - 1].y + 1
tableValues[x][a] != ' ' && a > -1
True
int temp = a - 1


tableValues[x][temp] != ' ' && temp > -1
True
False
temp--
temp++
temp < a
True
False
wordyword += tableValues[x]
temp
wordLocation[LocationIndex].x = x
wordLocation[LocationIndex].y = temp
LocationIndex++
temp++
wordyword += tableValues[x]
a
wordLocation[LocationIndex].x = x
wordLocation[LocationIndex].y = a
LocationIndex++

Testing and Validation:

During the process of making our project we encountered several problems and at most times we either solved the problems by fixing logics or by hit and trial.

Problem Specification:

Logical Errors:

- The biggest problem encountered was in setting the pixels of all the tiles and adjusting pixels of each tile with respect to the other, so we adjusted pixels in a ratio 36x36.
- The table pixels were not precise.
- In the initial stages of making the rack of tiles some key tiles appeared on the rank where as the rest either did not appear or appeared as blank tiles.
- A major problem was that tiles were not being moved and placed as required. Tiles from rack 2 were displaced as we moved tiles from rack 1.
- Some times the tiles did not move at all.
- We faced placement errors when the tiles did not move to the required position.
- During the process of trying to play scrabble tiles placed at the right bottom of the board were not adjusted properly.
- In the beginning we were unable to put the tiles on the rack with precision or to set them straight.
- The tiles were able to be picked, dragged but unable to be dropped.
- We encountered problems during the positioning of sprites.
- Another problem was that tiles were being stored but not displayed.

Future Developments:

Although we gave our best in making a scrabble game and provided all necessary availabilities and options and successfully implemented it. But the project still lacks a few details and implementations due to certain limitations. Some additions could be made in any future developments of our project are:

- Better Interface
- More Players
- Better Control Over Racks and Tiles

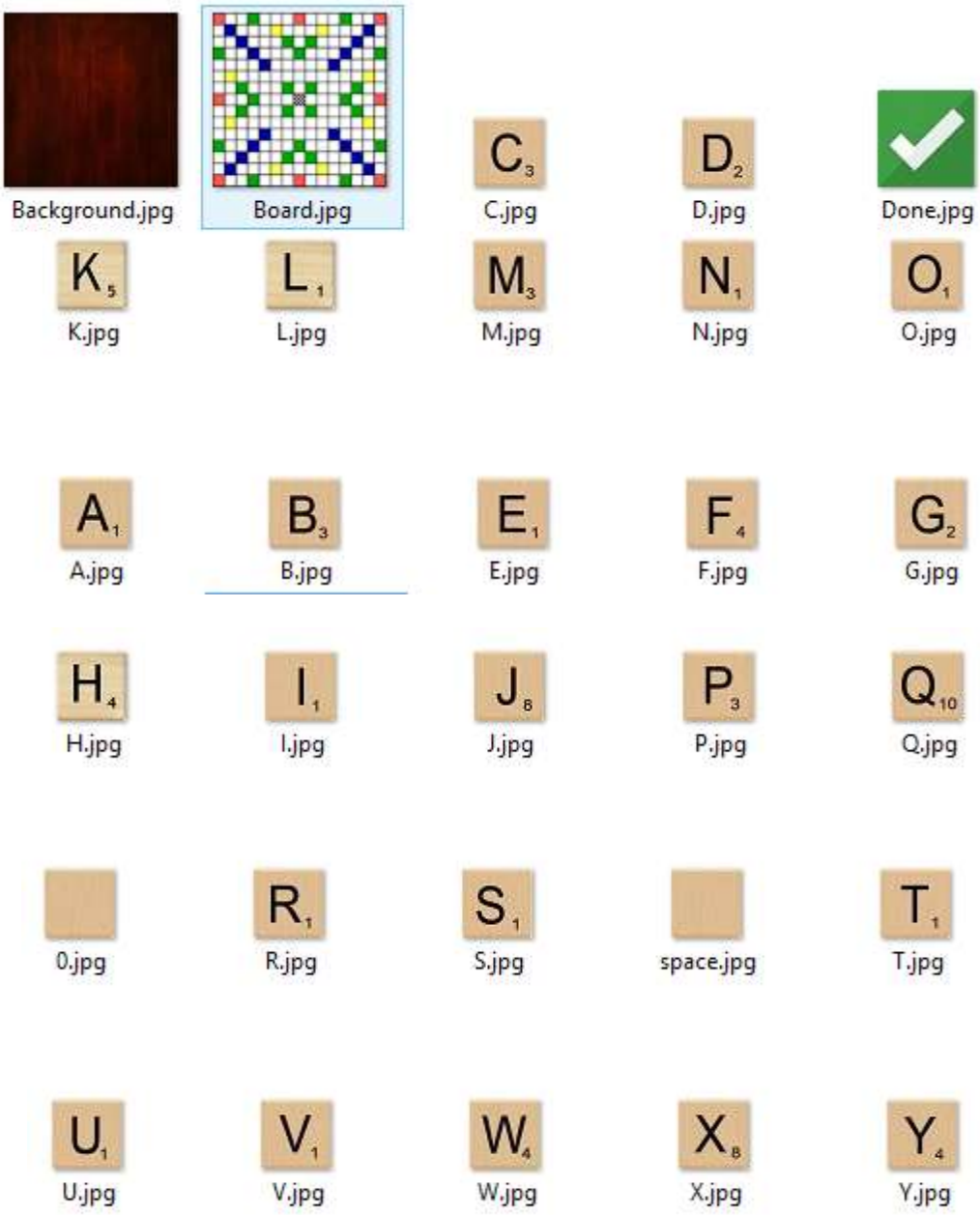
Unfortunately, we had to place certain limitations due to un availability of time.

Conclusion:

Our project, the computerized two players "Scrabble Game" provides a successful implementation of the game scrabble. Along with, all basic rules and regulations of the scrabble game and also provides and checks all sorts of restrictions and limitations in order to prevent the players from breaking any kind of rules. The game has been designed not only to increase the players vocabulary but also provides a challenge to your brain. Thus, changing the way you use your brain. Our project uses a subset of the dictionary containing twenty thousand words. The primary focus of the project is to prevent manual errors and cheating during the game and precisely checks and matches every word from the dictionary so that only valid words are made During the compilation of this report based on our project of Scrabble Game we wrote down an abstract and conclusion concerning the crux of our project. We highlighted the problems we encountered during the process of making the project along with the solutions. We provided information about the basics of our program and its elements. Additionally, we provided a detailed design of our program. We described the methods through which we tested and validated our program and also put forward our main strategies during development of code. We explained the short comings of our program resulting from certain limitations and the possible future developments. Furthermore, a detailed user manual and appendix has been provided with short description of all relevant terms, functions, data structures, libraries etch. The project thus provides a successful implementation of computerized Two Player Scrabble Game.

Preloaded Data:

Scrabble Board and Tiles Images:



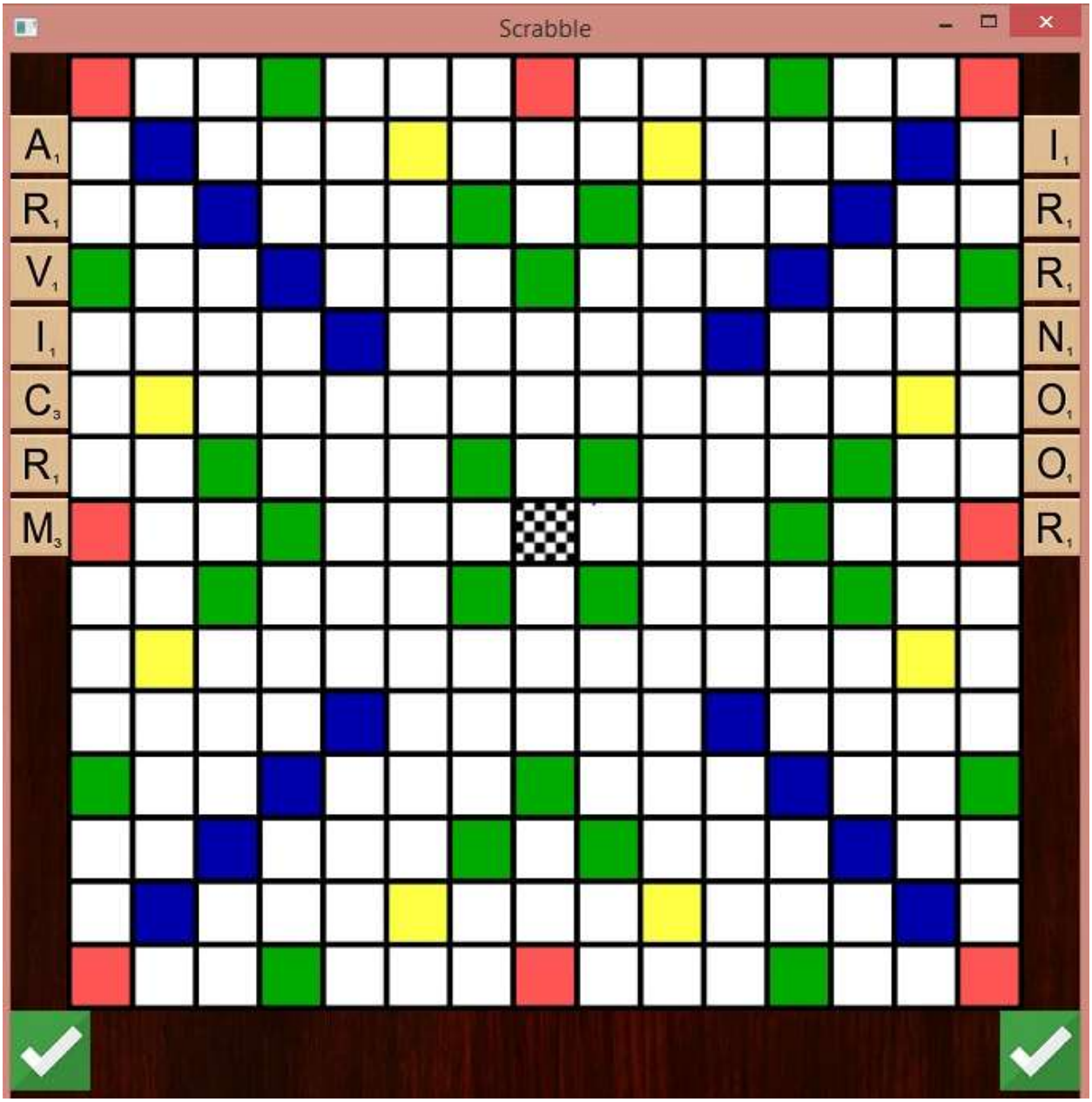
Dictionary Words:

File Edit Format View Help

```
ids
hugo
wagner
constraint
groundwater
touched
strengthening
cologne
gzip
wishing
ranger
smallest
insulation
newman
marsh
ricky
ctrl
```

```
infringement
bent
laos
subjective
monsters
asylum
lightbox
robbie
stake
cocktail
outlets
swaziland
varieties
arbor
mediawiki
configurations
poison
```

Console Output:



Appendix:

Standard Libraries:

A **standard library** in computer **programming** is the **library** made available across implementations of a **programming** language.

- <ctime>: This header file contains definitions of functions to get and manipulate date and time information.
- <iostream>: Header that defines the standard input/output stream objects.
- <string>: This header introduces string types, character traits and a set of converting functions.
- <Windows.h>: **windows.h** is a **Windows**-specific header file for the C and C++ programming languages which contains declarations for all of the functions in the **Windows** API, all the common macros **used** by **Windows** programmers, and all the data types **used** by the various functions and subsystems.
- <vector>: This file exports the Vector class, which provides an efficient, safe convenient replacement for the array type in C++.
- <time.h>: This header defines four variable types, two macros and various functions for manipulating date and time.

  Variables:

  1. Size_t: This is the unsigned integral type and is the result of the sizeofkeyword.
  2. clock_t: This is a type suitable for storing the processor time.
  3. time_t : This is a type suitable for storing the calendar time.
  4. struct tm: This is a structure used to hold the time and date.

  Macros:

  1. NULL: This macro is the value of a null pointer constant.
  2. CLOCKS_PER_SEC: This macro represents the number of processor clocks per second.
- <SFML\Graphics.hpp>: Simple and Fast Multimedia Library

Directives:

The first lines that begin the program are *directives*. The first is a *preprocessor directive*,
and the second is a using *directive*.

- ✓ **Preprocessor Directives:**

Preprocessor directives are lines included in a program that being with the character #, which make them different from a typical source code text. They are invoked by the compiler to process some programs before compilation.

- ✓ **using namespace sf::**

The built in **C++** library routines are kept in the standard **namespace**. That includes tools of SFML like mousebutton, event etc. Because these tools are **used** so commonly, it's popular to add "using **namespace sf**" at the top of your source code so that you won't have to type the **sf**:: prefix constantly

- ✓ **using namespace std::**

The built in **C++** library routines are kept in the standard **namespace**. That includes stuff like cout, cin, string, vector, map, etc. Because these tools are **used** so commonly, it's popular to add "using **namespace std**" at the top of your source code so that you won't have to type the **std**:: prefix constantly

- ✓ **#pragma comment(linker, "/SUBSYSTEM:windows/Entry:mainStartup")**:

When trying to display output in windows instead of console an error is generated. With the help of this preprocessor directive all errors are handle and output is only generated on required window.

Functions:

A **function** is a group of statements that together perform a task.

- main():The **main function** is called at program startup after initialization of the non-local objects with static storage duration.

- Getter function: Used to take input.
- Setter Function : Used to set the values of data members.

Arrays:

Array is a data structure which stores a fixed-size sequential collection of elements/variables of the same type. C++ allows multidimensional arrays i.e. 2D and 3D arrays.

Pointers:

A **pointer** is a programming language object, whose value refers to (or "points to") another value stored elsewhere in the computer memory using its memory address. A **pointer** references a location in memory, and obtaining the value stored at that location is known as dereferencing the **pointer**.

Class:

A **class** in C++ is a user **defined** type or data structure declared with keyword **class** that has data and functions.

- Constructor: A **constructor** is a kind of member function that initializes an instance of its class.
- Destructor: A **destructor** is a special member function that is called when the lifetime of an object ends.
- Objects: "object" refers t
- **No table of contents entries found.**o a instance of a class where the object can be a combination of variables, functions, and data structures.

File Handling:

- **File.** The information / data stored under a specific name on a storage device, is called a file.
- **Stream.** It refers to a sequence of bytes.
- **Text file.** It is a file that stores information in ASCII characters. In text files, each line of text is terminated with a special character known as EOL (End of Line) character or delimiter character. When this EOL character is read or written, certain internal translations take place.

- **Binary file.** It is a file that contains information in the same format as it is held in memory. In binary files, no delimiters are used for a line and no translations occur here.

Classes for file stream operation:

- **ofstream**: Stream class to write on files.
- **ifstream**: Stream class to read from files**.**
- **fstream:** Stream class to both read and write from/to files.

File mode parameter	Meaning
ios::app	Append to end of file
ios::in	open file for reading only
ios::out ios::left	open file for writing only the output is padded to the *field width* appending *fill characters* at the end.

Exceptions:

A C++ **exception** is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. **Exceptions** provide a way to transfer control from one part of a program to another. C++ **exception** handling is built upon three keywords:

- throw: A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- catch: A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
  The **catch** keyword indicates the catching of an exception.
- try: A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Escape Sequences:

An escape sequence is a sequence of characters that does not represent itself when used inside a character or string literal, but is translated into another character or a sequence of characters that may be difficult or impossible to represent directly.

- \t : Horizontal tab.
- \n : line feed - new line
- \: Single quote.
- \":Double quote.
- \\: backslash/Comment

Data types:

Based on the **data type** of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

- Time_t: The **time_t** is a data type defined for storing system time values.
- Int: **int** is a fundamental type built into the compiler and used to define numeric variables holding whole numbers.
- char: it is a powerful data type used to handle and manipulate strings of **characters.**
- float: **float** is short for "**floating** point." It is a fundamental type built into the compiler and used to **define** numeric values with decimal points, which accommodate fractions.
- string: The term **string** generally **means** an ordered sequence of characters.
- bool: The Boolean data type is used to declare a variable whose value will be set as true (1) or false (0). To declare such a value, you use the **bool** keyword. The variable can then be initialized with the starting value. A Boolean constant is used to check the state of a variable, an expression, or a function, as true or false.

Statements:

A **statement** is a block of code that does something.

- if…else: The **if else statement** allows you to control **if** a program enters a section of code or not based on **whether** a given **condition** is true or false.

- system("pause"): It allows the program to wait until user hit enter so they can see their output on console.
- system("cls"): It is used to clear the screen in Visual C++.
- break: When the **break statement** is encountered inside a **loop**, the **loop** is immediately terminated and program control resumes at the next **statement** following the **loop**.Syntex:break;
- continue: The **continue** statement works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.Syntax: continue;
- cin.get(): **cin**.**get()** forces the program to wait for the user to enter a key before it can close, and you can see the output of your program.
- getline: **getline** will read from the target IO Stream until a new-line character is found and store the result in the variable
- cout: Standard output stream..
- cin: Standard input stream.
- or(||):Executes a statement if either condition is satisfied.
- and(&&):Executes a statement if both conditions are satisfied.

Loops:

In computer **programming**, a **loop** is a sequence of instruction s that is continually repeated until a certain condition is reached.

- For-loop: A for-**loop** is a control flow statement for specifying iteration, which allows code to be executed repeatedly.
- While-loop: A **while loop** is a control flow **statement** that allows **code** to be executed repeatedly based on a given Boolean condition.
- Do While-loop: A **do while loop** is a control flow statement that executes a block of code at least once, and then repeatedly executes the block depending on a given Boolean condition at the end of the block.

Operators:

An **operator** is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

- (::)Scope Resolution Operator
- (>>)Extraction Operator
- (<<)Insertion/Cascading Operator
- (%)Remainder/modulus operator
- (?:)Conditional/Ternary Operator

C++ is rich in built-in **operators** and provides the following types of **operators**:

- Arithmetic Operators

Operator	**Description**
+	Adds two operands
-	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by de-numerator
%	Modulus Operator and remainder of after an integer division
++	**Increment operator**, increases integer value by one
--	**Decrement operator**, decreases integer value by one

- **Relational Operators**

Operator	**Description**

==	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

- **Logical Operators**

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.

||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.

- **Assignment Operators**

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add and assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract and assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply and assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide and assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus and assignment operator, It takes modulus using two operands and assign the result to left operand

- **Misc Operators**

Operator	Description
Condition ? X : Y	**Conditional operator**. If Condition is true ? then it returns value X : otherwise value Y
,	**Comma operator** causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
. (dot) and -> (arrow)	**Member operators** are used to reference individual members of classes, structures, and unions.
&	**Pointer operator &** returns the address of an variable. For example &a; will give actual address of the variable.
*	**Pointer operator *** is pointer to a variable. For example *var; will pointer to a variable var.

References:

- https://www.sfml-dev.org/
- https://en.wikibooks.org/wiki/Scrabble/Rules
- https://en.wikipedia.org/wiki/Scrabble
- https://www.google.com.pk/search?q=scrabble+keys&source=lnms&tbm=isch&sa=X&ved=0ahUKEwigorKo4rzYAhVO6KQKHWqDCqUQ_AUICigB&biw=1242&bih=602#imgrc=lBh6d_DO9KbrBM:
- https://lms.nust.edu.pk/portal/mod/resource/view.php?id=417691