Report

# Methods

In the **StartGameForm** class there are two methods that were added to reduce the size of the class. Those are:

**SaveButtonClick(***…***)**

It takes in as parameters the player number, bool indicating that the player has saved as a ref, bool indicating that the opponent has saved, and all the GUI components related to player 1 that need to be customised. This includes the radio buttons to select the dragon type which will be used to check which dragon type is selected.

It is called on **Lines 68** and **74**. Which are withing the methods called when player 1 and player 2 save respectively.

**UpdateColor(***…***)**

It takes in the background colour, foreground/text colour and a params array of GUI components as parameters. It then goes through all the GUI components with a for-each loop and sets their background and foreground colours to those specified in the parameters.

It is called on **Lines** **88-90**, **96**, **102**, **108**, **114**, **120**, **126**, **132** and **138**.

# Conditional/Ternary Operators

A conditional/ternary operator works as follows:

*[data type]* variable = *[condition to be tested]* ? *[value if true]* : *[value if false]*;

It can be written with an if-else statement as follows:

*[data type]* variable;

if (*[condition to be tested]* == true)

{

variable = *[value if true]*;

}

else

{

variable = *[value if false]*;

}

It can be found on **Line 84** in the **StartGameForm** class. It is a bit complicated so I will explain it with if-else statements.

It says:

string type = rbtnFireDragon.Checked ? FIRE\_DRAG\_NAME : rbtnIceDragon.Checked ? ICE\_DRAG\_NAME : rbtnWindDragon.Checked ? WIND\_DRAG\_NAME : rbtnEarthDragon.Checked ? EARTH\_DRAG\_NAME : "";

Written with if-else statements:

string type;

if (rbtnFireDragon.Checked)

{

type = FIRE\_DRAG\_NAME;

}

else

{

if (rbtnIceDragon.Checked)

{

type = ICE\_DRAG\_NAME;

}

else

{

if (rbtnWindDragon.Checked)

{

type = WIND\_DRAG\_NAME;

}

else

{

if (rbtnEarthDragon.Checked)

{

type = EARTH\_DRAG\_NAME;

}

else

{

type = “”;

}

}

}

}

It can be found on **Lines 51-52**, **54**, **64**, **75-76**, **78**, **88**, **99**, **126** and **154-156** in the **TurnPlayerForm** class. With **Lines 64** and **88** being unique, but following the same format, and the rest following the basic format.

**Lines 64** and **88** are written as follows:

blockMessage = (playerTurn == 1 ? p2IsBlocking : p1IsBlocking) ? " blocks it and" : "";

Written with if-else statements:

if (playerTurn == 1)

{

if (p2IsBlocking)

{

blockMessage = “ blocks it and”;

}

else

{

blockMessage = “”;

}

}

else

{

if (p1IsBlocking)

{

blockMessage = “ blocks it and”;

}

else

{

blockMessage = “”;

}

}

Dictionaries

Dictionaries were added to the **StartGameForm** class to add a fast way to get data based off the dragon type.

**Lines 42-49** link the dragon type to its picture. It is used on **Lines 148** and **149** to save the picture of the selected dragon type to the **TurnPlayerForm** form.

**Lines 51-58** link the dragon type to an array of its values. It is used on **Line 85** to get the values of the selected dragon type to parse as arguments to the **SaveValues(***…***)** method on **Line** **86**.

Properties

**Lines 12 and 14** in the **TurnPlayerForm** class are public properties that return the picture boxes for player 1 and player 2’s dragons respectively. They replace the **GetPicPlayer1Dragon()** and **GetPicPlayer2Dragon()** methods submitted in Part 1.

Lambda

The lambda function on **Line 39** in the **TurnPlayerForm** class replaces the method that was used in Part 1 to force the application to close when the form is closed. This is done because the method contained only one line and never needed to be removed from the **FormClosing** delegate, so a lambda worked perfectly fine and reduced the size.

# Other Changes

On **Lines 156-168** in the StartGameForm class I set the arrays to be new arrays that contain the values in the parameters rather than saving them to individual indices in the arrays that was seen in the pseudocode for Part 1.

On **Lines 62-63**, **86-87**, **101-102**, **104-105**, **157-158** in the **TurnPlayerForm** class, rather than checking if it is the players turn and setting the bools accordingly, the bools are always set according to the player turn **AND**/**OR** (depending on the bool) the bool’s current state.