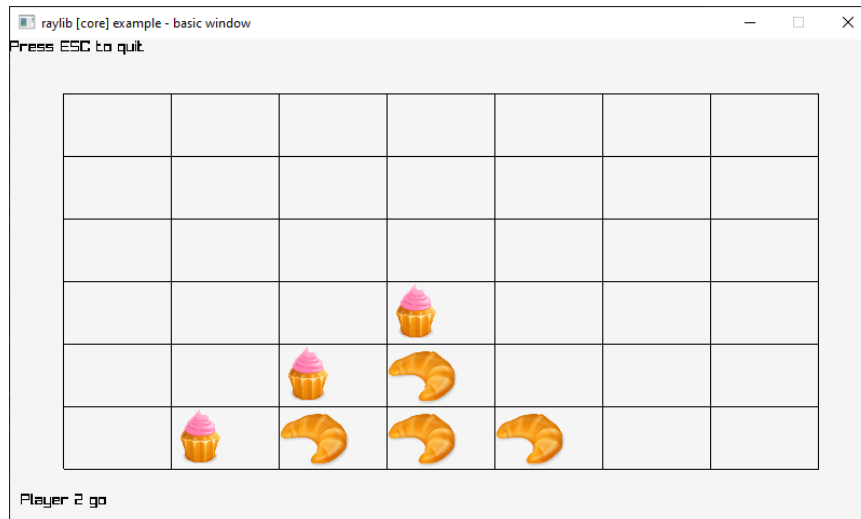


Tutorial – Algorithms

Introduction:

This this tutorial we are going to practice our algorithm writing skills by re-creating the game “Connect Four”.



Requirements:

For this tutorial, we are going to be using the *Connect Four* project code, available in the *AIE Year 1 Samples* repository <https://github.com/AcademyOfInteractiveEntertainment/AIEYear1Samples>.

- Download the Git repository and open the solution. Select the *CDDS_Algorithms* project as the Startup Project.

The *Connect Four* project is implemented using the *Raylib* framework.

Check for Win:

This project implements everything except the logic to determine if a player has won. This is the algorithm we will be creating in this tutorial.

Open the project in Visual Studio and review the code. Make sure that the project can compile and execute.

In its current form, you should be able to click on any square on the board and see either the cupcake or croissant player icon. You can click on as many tiles as you like, but at the moment there is no logic to determine if one of the players has won the game.

The *checkForWin()* function should contain the logic to determine if a win condition has occurred.

We'll start by writing a plain English description of what this function should do.

Function: Check For Win

Iterate over each tile on the board.

For each tile on the board, we will check whether it is the first tile in a series of four tiles of the same.

If the current tile is empty, move onto the next tile.

Check each of these eight directions, starting on the current tile: left, right, up, down, diagonally up and left, diagonally up and right, diagonally down and left, and diagonally down and right.

If any of these checks is true, return true from the function. Otherwise return false.

Read through the above description of our function. What we've described is what is called a 'brute force' approach.

In a brute force algorithm, we make no attempt to optimize our solution according to the characteristics of the problem.

Clearly there are some things we could have done here to make our algorithm more efficient. For example, if we are starting on the first tile in the bottom-left corner then obviously we cannot make a winning match by going down or left (since those tiles are off the board). But in the algorithm above, we still do those checks.

Working in teams or individually, attempt to write a better description that will minimize the number of checks we need to make when looking for a winning condition.

After you have a plain English description you are happy with, you can turn that into pseudo-code (or you might be comfortable skipping the plain English step and going directly to writing pseudo-code).

The pseudo-code for the algorithm above can be written as follows:

```
Function: checkForWin()  
  For each tile on the x axis  
    For each tile on the y axis  
      Get the tile at location x,y  
      If the tile is empty, continue to the next tile  
      If the next 3 tiles to the right match the current  
        tile, return true  
      If the previous 3 tiles to the left match the current  
        tile, return true  
      If the 3 tiles above the current tile match the  
        current tile, return true  
      If the 3 tiles below the current tile match the  
        current tile, return true  
      If the 3 tiles diagonally to the right and down match  
        the current tile, return true  
      If the 3 tiles diagonally to the left and down match  
        the current tile, return true  
      If the 3 tiles diagonally to the right and up match  
        the current tile, return true  
      If the 3 tiles diagonally to the left and up match  
        the current tile, return true  
  
  return false
```

Our pseudo-code is a bit more verbose than our plain English description, and you should now see how many checks we're actually doing for each tile on the board.

If you or your team came up with a different algorithm, create pseudo-code for your algorithm.

On occasion it is helpful to turn our algorithm into a flow chart. Some people find drawing or reading a diagram more helpful than the corresponding pseudo-code or plain English description, and there is no reason why we cannot use a flow chart to describe our program logic.



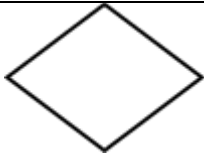
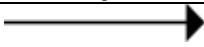
A good program to use for drawing flow charts (and other diagrams) is a website called <https://www.draw.io/>

This site will allow you to save your diagrams to your Google Drive or Dropbox folder, or download them to your computer in xml format.

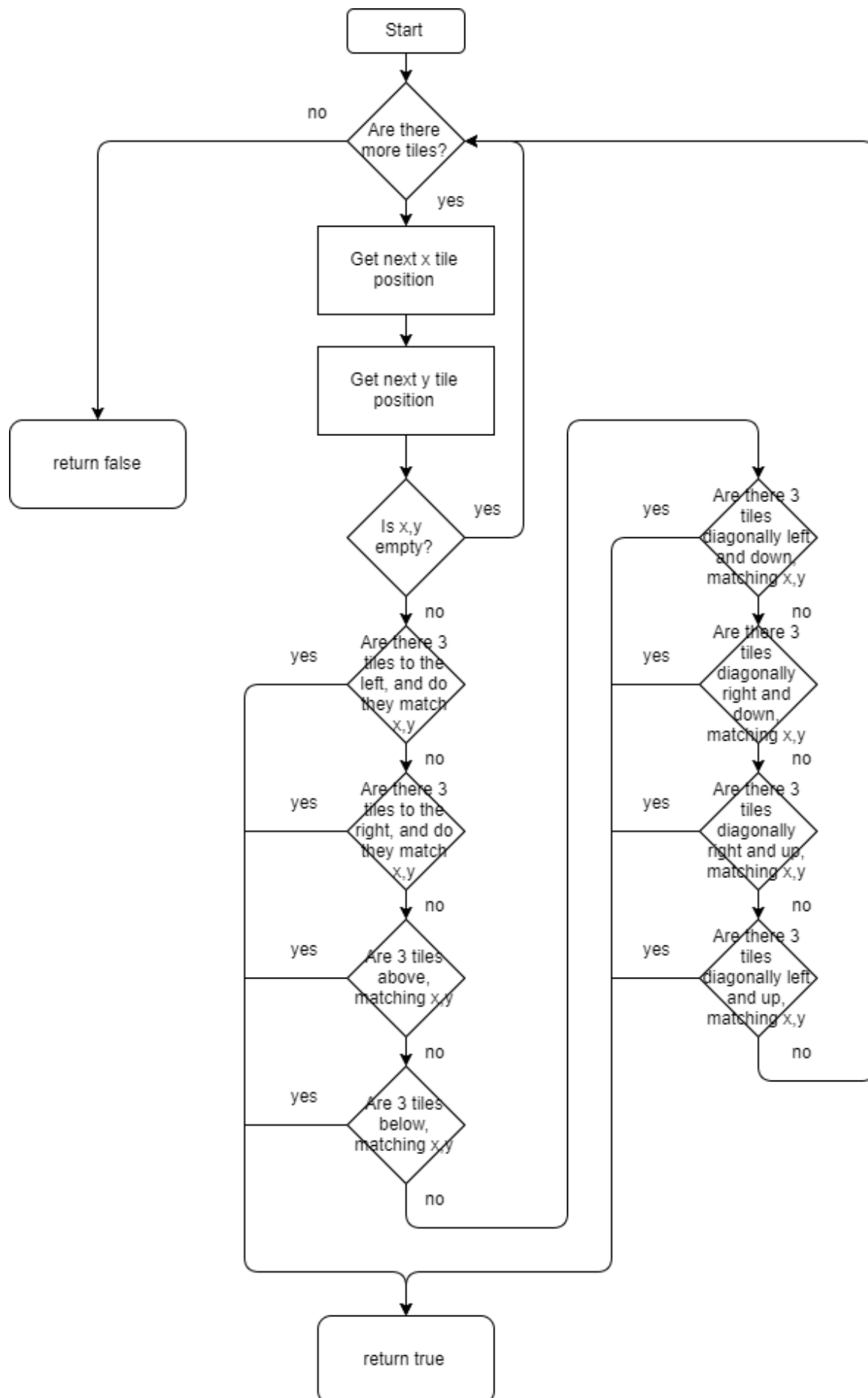
Please note that if you are drawing diagrams on this site for your assessments, you should export them to images and place them in a Word document when submitting your assessment. (The saved xml files from this site are not appropriate to use for your assessment submissions).

Go to the website and start drawing a flow chart using the shapes in the *General* tab.

Flowcharts contain only a few shapes and so are easy to draw, but each shape has a meaning so be sure to use the correct on.

	Use a rectangle with rounded corners for terminal nodes – the entry point of your program or function (i.e., where the function starts, and where it ends).
	This is a 'process' rectangle. It represents some operation to be performed.
	Yes/No or True/False conditions are indicated with a diamond. If you want to make a <i>switch</i> statement, use several diamonds in a series.
	All nodes are joined with these arrows. Arrows indicate the flow of execution of the program.

The flowchart for our *checkForWin* algorithm can be drawn as shown on the next page.



If you have written your own algorithm, use the *draw.io* site to create a flow chart of your algorithm.

When drawing flow charts, it is often much quicker to sketch out a quick diagram with a pen and paper first, then transfer your final design into a program like *draw.io*.

Challenge:

Now that you have an algorithm for the *checkForWin()* function, implement this algorithm in code.

If you have not created your own algorithm, implement the one from this tutorial.

Once you have finished, test your implementation by playing a few games of *Connect Four*. Be sure to test as many possible cases for the win condition as you can think of.