Create Performance Task Template 2018

2a.

This is my Relax app. The purpose of my app is to make it easier for people to put their phones down and become more productive. The language I use to program my app is called blockly. The first thing the user will see when entering into my app for the first time in the day is a prompt asking them what their goal for the day is. To help the user put their phone down I created a mini game within my app. This game works by having the user enter the time of how long they want to put their phone down. Now once I've put my time in the app will display a timer with a message explaining the point of the game. When the user goes to leave the screen it will tell them that they have killed the tree they were trying to keep alive.

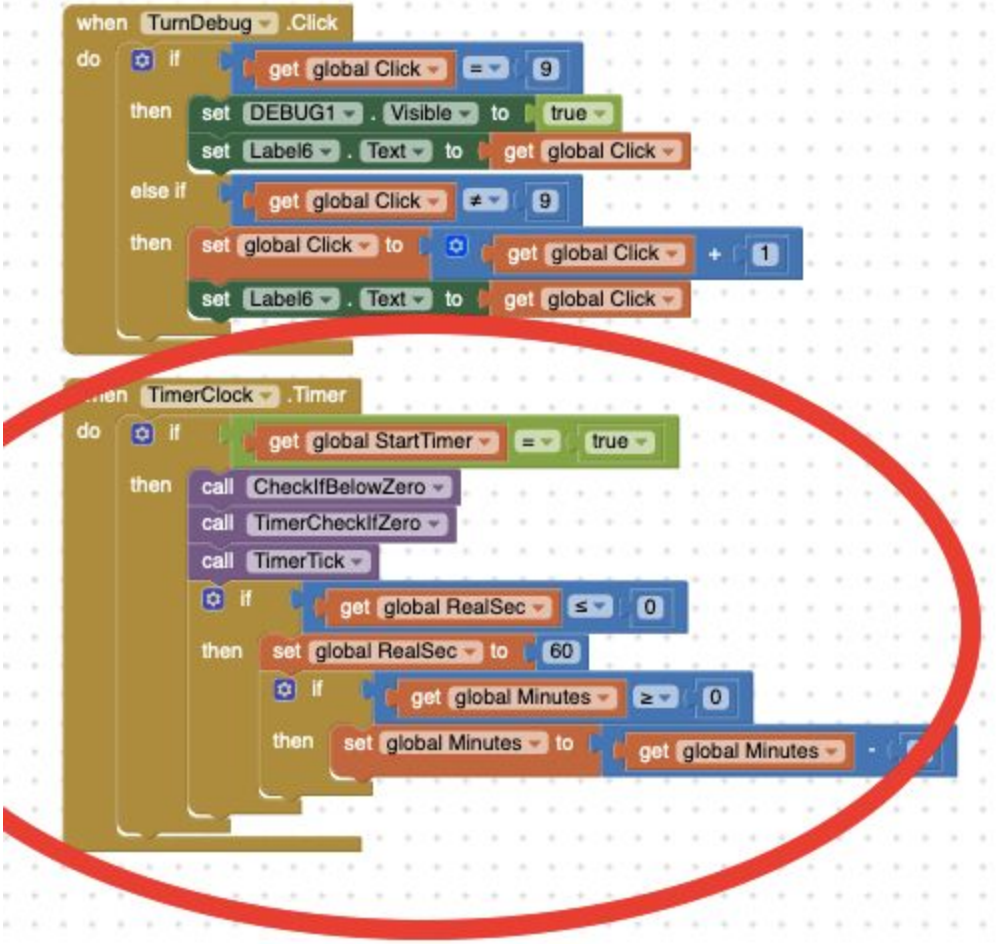
2b.

One of the difficulties I faced while creating my app was that my timer's seconds and minutes would drop into the negative numbers. This caused my app to not function correctly and stopped one of the main features of my app to work. I started off by testing out the app and identifying which parts of the code were activating when the bug was occurring. One of the ways I did this was I created a debug console to display what part of my code was running. This allowed me to realize that the checks for negative numbers were not running in my code. I fixed this bug by adding multiple checks to my code. After testing this allowed my app to work correctly. Another difficulty I ran into was that my app would crash after 30 minutes of continued use. After long testing session I was able to track this problem back to my timer code. The problem was that my variable to track the amount of cycles was exceeding the limit that I put in for debug purposes to fix earlier problems. After removing this code my app would no long crash after 30 minutes.

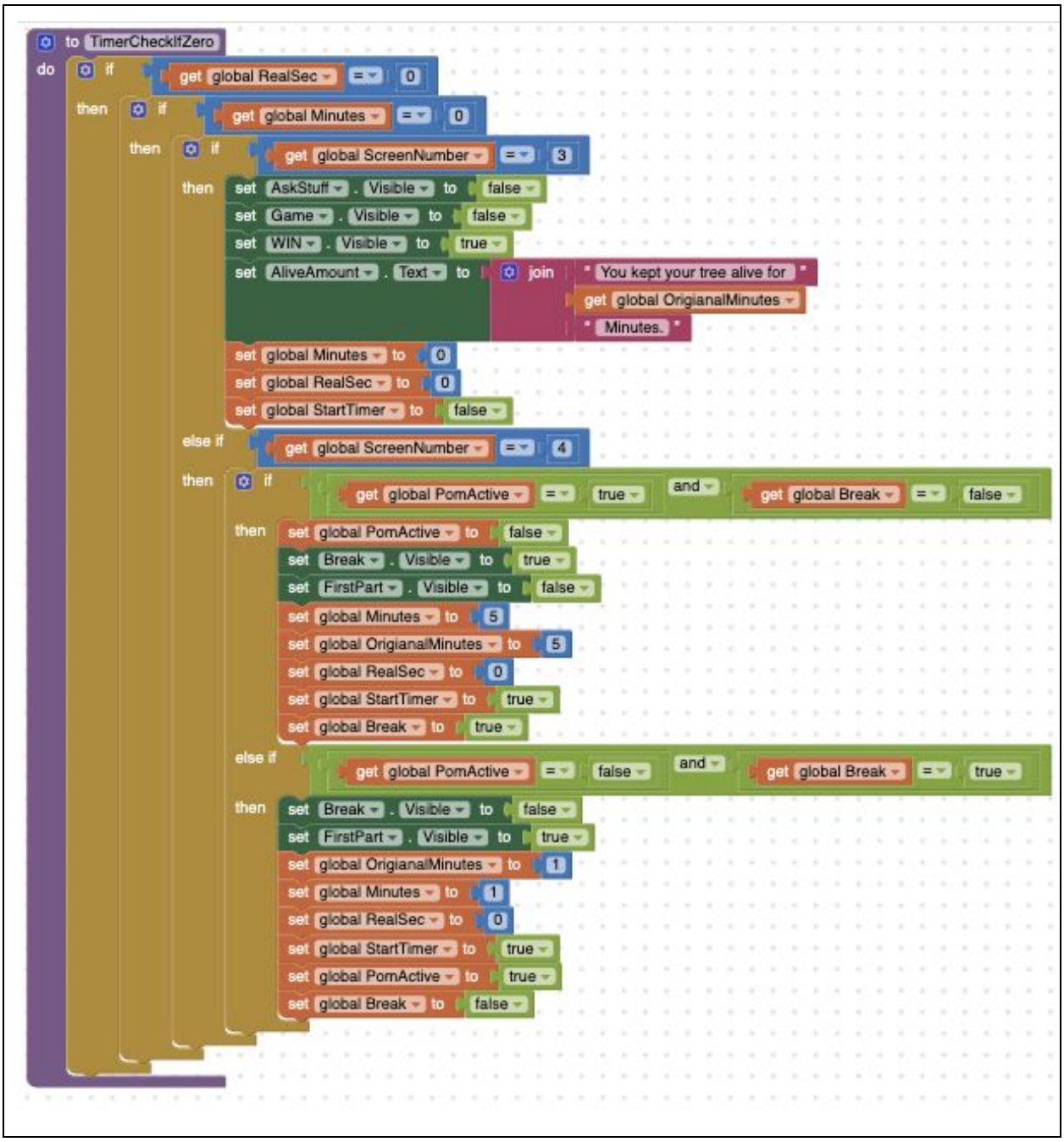
2c.

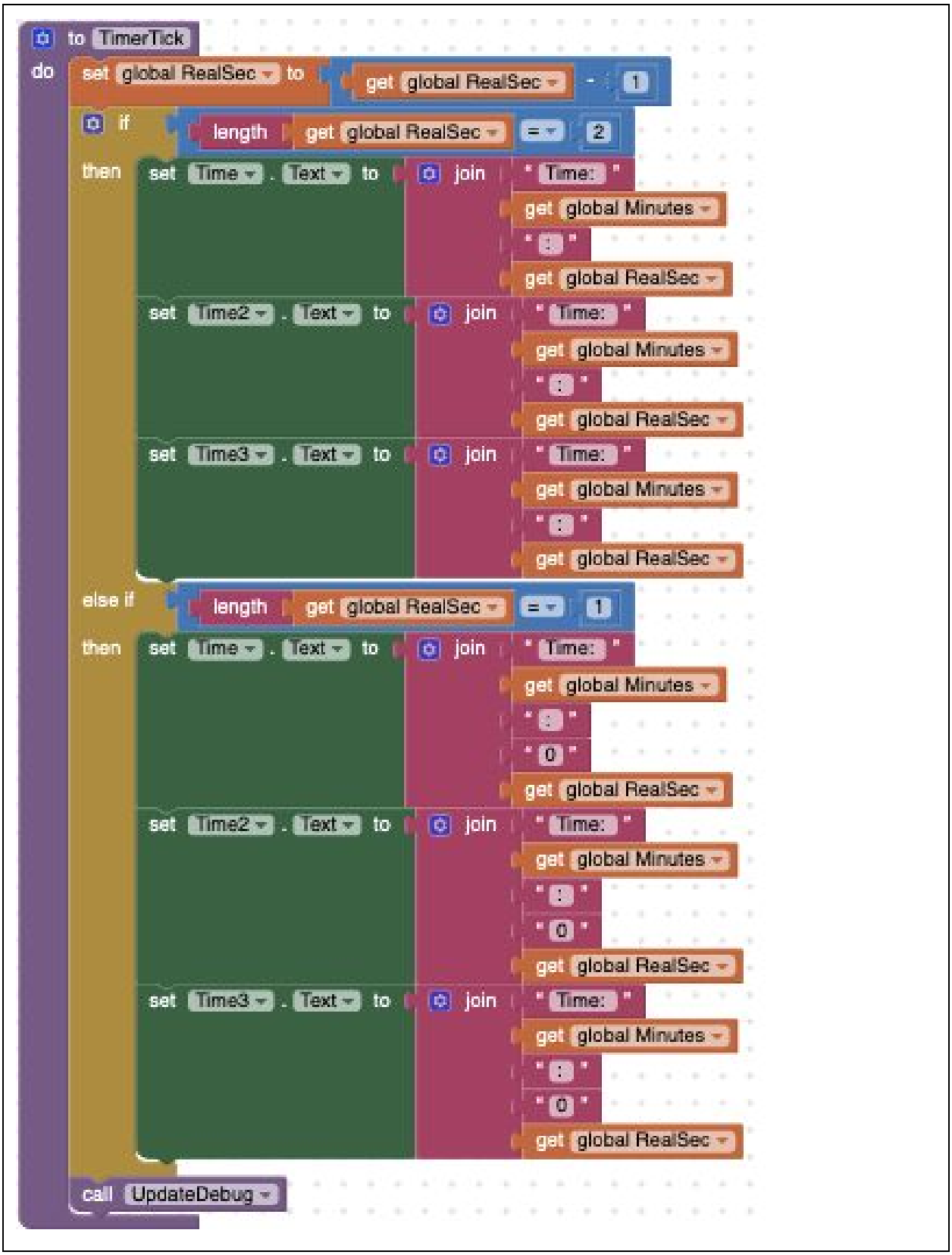
The algorithm I created is called TimerClock. The intended purpose of this algorithm is to create a countdown clock. My algorithm starts off by running every second, when it runs it checks to see if the variable StartTimer is equal to true. It then calls the three other algorithms, CheckIfBelowZero, TimerCheckIfZero, and TimerTick. Inside of CheckIfBelowZero there is a math statement to check if RealSec is equal to -1. After that it checks if minutes = -1. The point of this algorithm is to handle when the timer breaks and goes below zero. The next algorithm is TimerCheckIfZero, this algorithm uses the math statement Realsec = 0 and minutes = 0. If both of these statements are true the algorithm will run the code to show the user the win screen or the next screen. The final algorithm is TimerTick. TimerTick subtracts

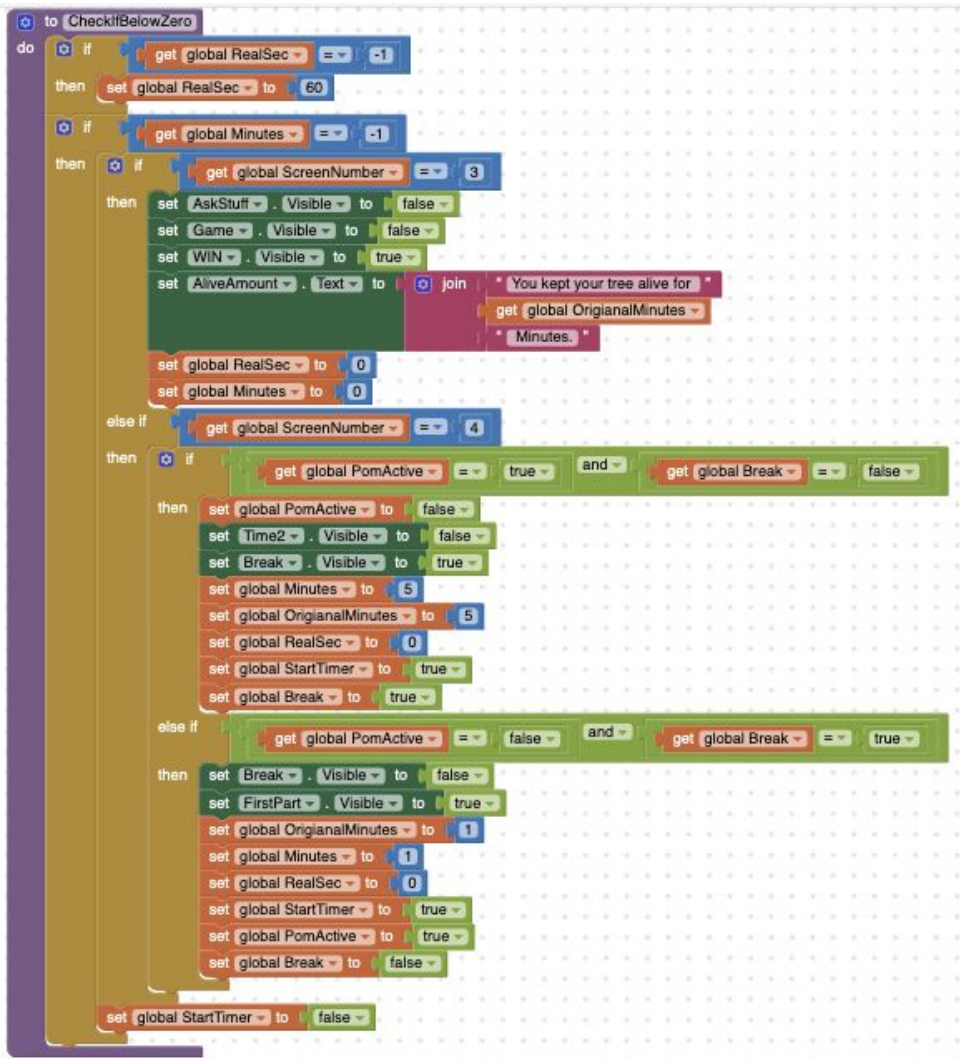
one from the seconds and then updates the text of the timer. After all three of these algorithms run the TimerClock algorithm will then check if RealSec is greater than or equal to 0 if so it will set RealSec to sixty and then check if minutes >= 0, if this is true it will subtract 1



from minutes.


to TimerCheckIfZero
do if get global RealSec = 0
then if get global Minutes = 0
then if get global ScreenNumber = 3
then set AskStuff . Visible to false
set Game . Visible to false
set WIN . Visible to true
set AliveAmount . Text to join " You kept your tree alive for "
get global OrigianalMinutes
" Minutes. "
set global Minutes to 0
set global RealSec to 0
set global StartTimer to false
else if get global ScreenNumber = 4
then if get global PomActive = true and get global Break = false
then set global PomActive to false
set Break . Visible to true
set FirstPart . Visible to false
set global Minutes to 5
set global OrigianalMinutes to 5
set global RealSec to 0
set global StartTimer to true
set global Break to true
else if get global PomActive = false and get global Break = true
then set Break . Visible to false
set FirstPart . Visible to true
set global OrigianalMinutes to 1
set global Minutes to 1
set global RealSec to 0
set global StartTimer to true
set global PomActive to true
set global Break to false


to TimerTick
do
set global RealSec to
get global RealSec - 1
if length get global RealSec = 2
then
set Time . Text to join "Time:" get global Minutes ":" get global RealSec
set Time2 . Text to join "Time:" get global Minutes ":" get global RealSec
set Time3 . Text to join "Time:" get global Minutes ":" get global RealSec
else if length get global RealSec = 1
then
set Time . Text to join "Time:" get global Minutes ":" "0" get global RealSec
set Time2 . Text to join "Time:" get global Minutes ":" "0" get global RealSec
set Time3 . Text to join "Time:" get global Minutes ":" "0" get global RealSec
call UpdateDebug



2d.

The abstraction that I selected is the code I use to control what view is visible to the user. My code starts off by checking what screen is wanted to be made visible. It does this by using logic to determine the the wanted screen. First it starts off by checking if screen number is less then or equal to 0. If this is true it will set the screen number to 1. Next my abstraction checks a sequence of different logic statements to see what screen it wants to switch to and display to the user. When it finds the screen that it wants to display it will set the correct views to visible or invisible. At the end of the program it will then call another procedure to update my debug screen. My code segments can be classified as an abstraction because I can call it from anywhere in the code by calling the procedure and supplying the Desired Screen

number. This allowed my code to be less complex by not crowding up all the screen switching buttons with massive amounts of repeated code.