ESOF 2571 Project Report

Nicholas Imperius: 0645031, Jimmy Tsang: 1098204, Kristopher Poulin: 0883832

Lakehead University

Thunder Bay, Ontario

Dr. Abdulsalam Yassine

I. Table Of Contents

II. Project Objective

A. Project Vision

The vision for this project is to develop a program that will simulate the popular arcade game Stacker. Stacker is a popular arcade game where there is a block that moves side to side and the goal is to stack the blocks on top of each other creating a taller stack that can reach the top of the screen. We intend to develop the program using the python programming language in conjunction with a Raspberry Pi and Sense HAT emulator.

B. Project Goal

The goal of this project is to develop our own small-scale version of Stacker that runs off a Raspberry Pi and Sense HAT. Moreover, our group will be including additional features which the original game does not include. These additional features consist of two new game modes, time trial and endless mode, in conjunction with the normal game mode. Lastly, arcade Stacker machines are manipulated to only allow users to win after a set profit amount has been achieved, our program will have no such unethical feature.

III. Equipment Used

A. Hardware Used

The development of the project occurred during the COVID-19 pandemic, thus, we were unable to use a physical Raspberry Pi or Sense HAT. Hence, we decided to emulate these hardware devices using a virtual machine instead. The function of the Raspberry Pi is to independently run the program and the purpose of the Sense HAT is to display the blocks and allow users to stack the blocks using the joystick, as you can see in the image below.

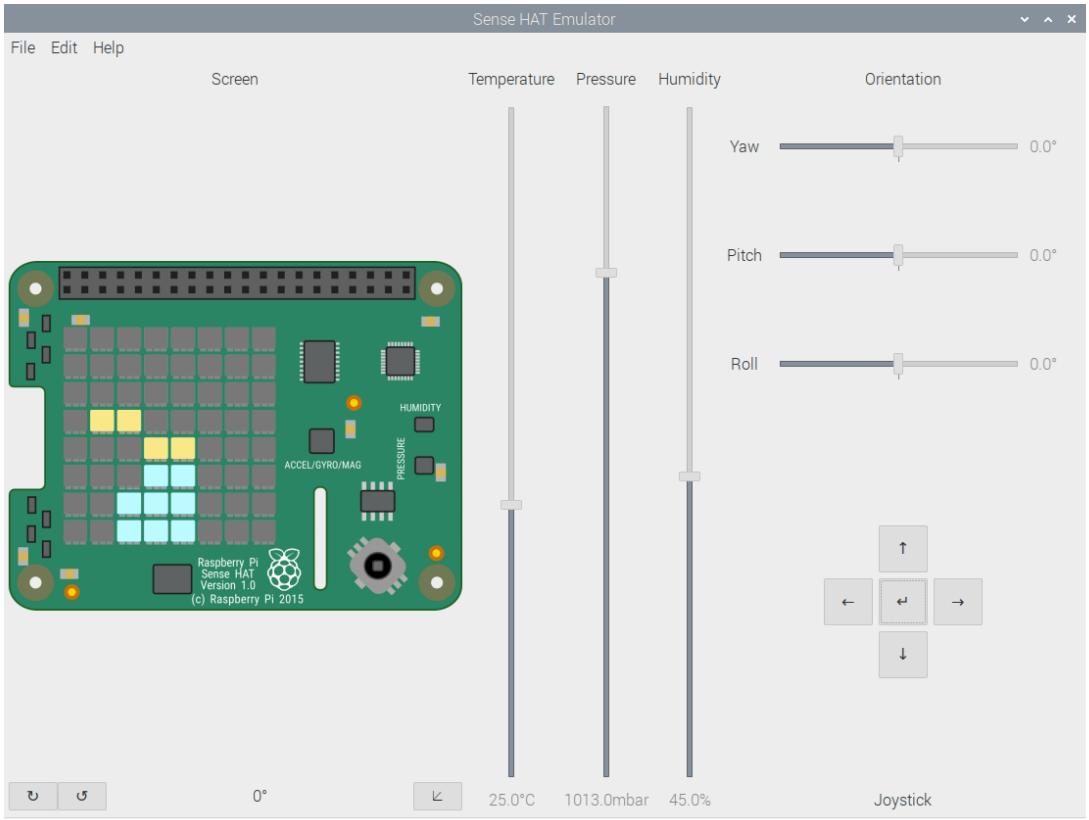

Figure 1: Sense Hat emulation program

B. Software Used

The software used during the development process of our program includes Python and Oracle VM VirtualBox which was running Raspbian OS virtual machine. Python was the programming language that was decided to be used, and the purpose of using the Raspbian operating system was to emulate the Raspberry Pi which includes the Sense HAT emulator.

However, we needed a virtual machine to run Raspbian, therefore, we determined that Oracle VirtualBox would be a good choice. As you can see in figure 2, this setup was fully functional.

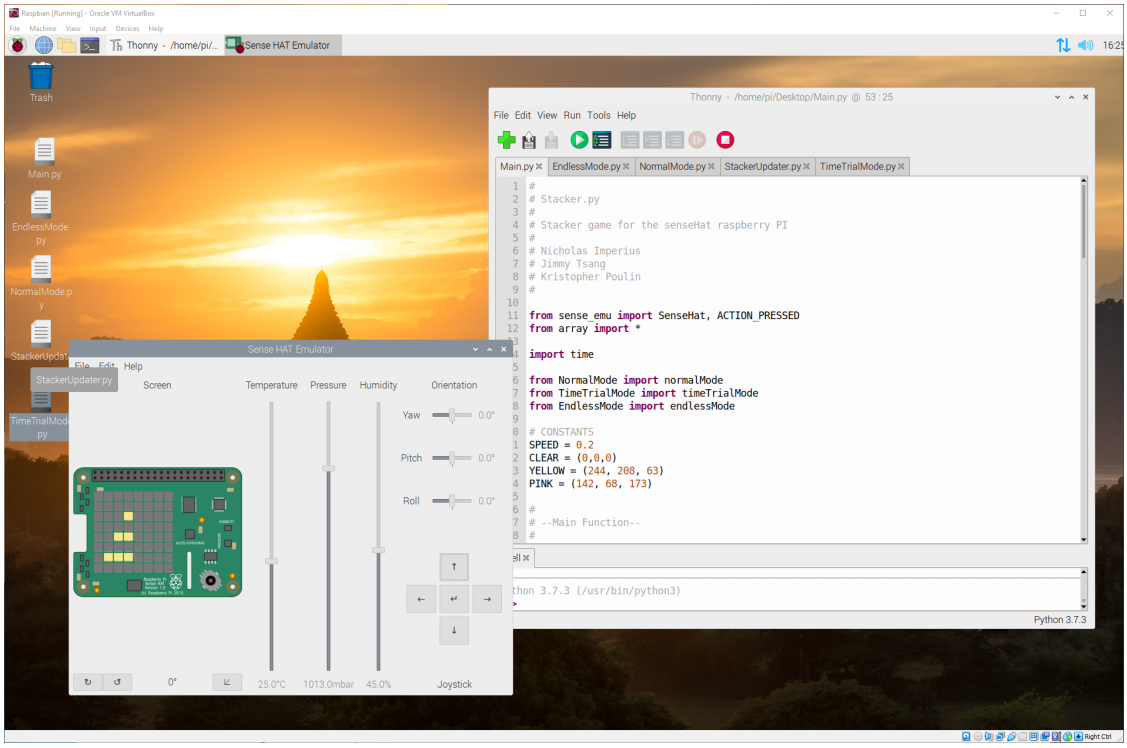

Figure 2: A screenshot of Oracle VM running Raspbian OS in order to program the Raspberry Pi

IV. Sense HAT Diagram

A. Diagram

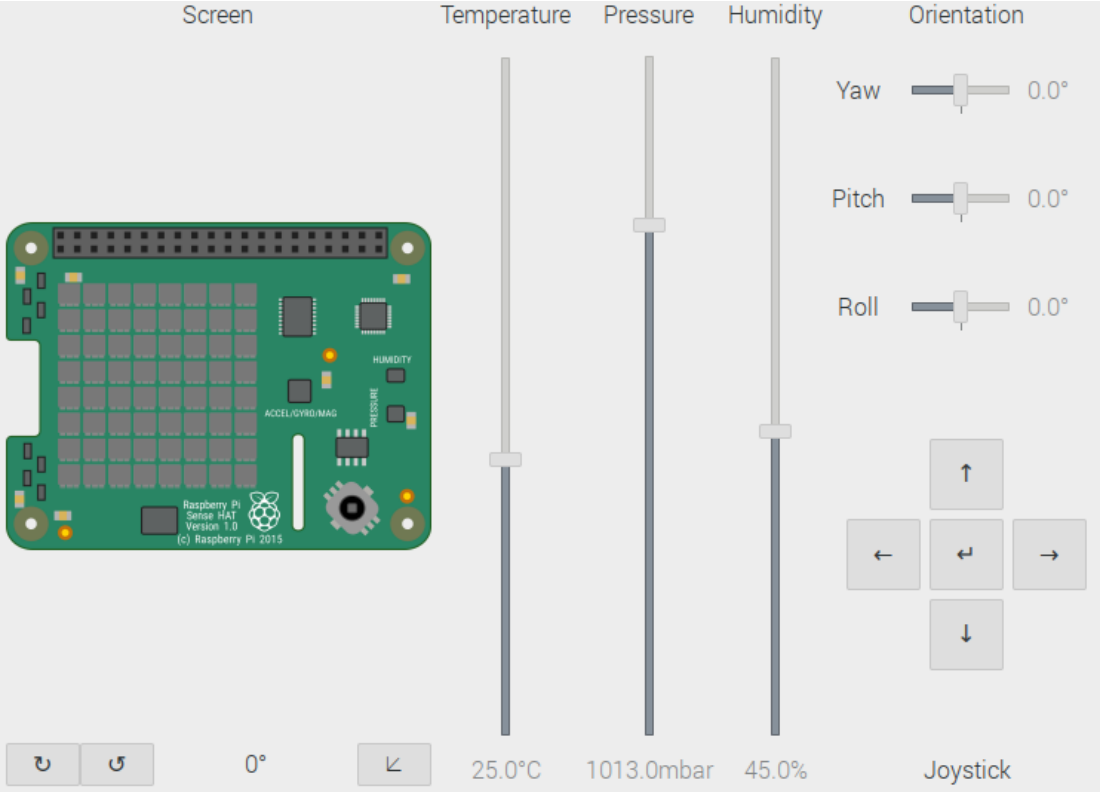

Figure 3: Diagram of Sense HAT Emulator

B. Description

Figure 3, above in Section A, shows the layout of the Sense HAT board in the Oracle VM VirtualBox emulator. The middle of the Sense HAT circuit board layout shows a 8 x 8 grid of dark grey blocks which are pixels that we can individually control the colour of. On the right side of figure 3 are all the controls that we would have on a real physical unit, but unfortunately, due to the global pandemic, we were not able to have one. The joystick controls are how the user will mainly interact with the module when playing the game. The up and down button of the joystick will be used to scroll to the desired game model whereas the middle button will be used to confirm the user's game mode selection as well as stack the blocks.

V. Procedure

A. Developmental Steps

The first step in developing our program was to decide on what the graphical user interface would look like. With the limited space available on the Sense HAT LED screen, we had to be creative with how everything was laid out. In the end, we decided to display a certain number of blocks that would relate to each game mode, one block for normal mode, two blocks for the time trial, and three blocks for endless as seen in figure 2 in Section III. The user would then select the game mode they wanted by using the joystick controls, once they confirm the mode a message will scroll across the screen letting them know the mode they are entering followed by a motivating 'GO!'. We would have liked to display the names of each of the game modes, but we were unable to do so with the limited screen size, and adding more scrolling text would have been time-consuming for the user resulting in a bad user experience.

The next step during development consisted of how we were going to make the blocks move and capture that information. We decided upon representing the entire screen in a 2-D array and keeping a 1-D array for the current row which we called stackedArray and currentX respectively. In each iteration of the loop the currentX would be updated and translated into the appropriate index of stackedArray so when we went to update the screen we knew what needed to be changed. Ultimately, we concluded that by using the 2-D and 1-D arrays we were able to enhance the speed of the program as well as make computing any other calculations more efficient.

The last step of developing our normal mode involved incorporating vertical movement of the blocks and object detection. To incorporate vertical movement, we used the 2-D array that represents the whole board and we would update values in it and compare it to the current board. Every time the middle button was clicked, we captured that information and would increment the row value. If there were any blocks that were not on top of another block, we would compare the blocks of the current row to the last and remove the outlying block. In addition, we also incorporated adjustment of the block speed and colour and the automatic removal of blocks after a certain row height was reached. If the user manages to make it to the top and successfully

stacks the last block, the program would verify that it is a win or loss by again checking the row below to see if there is at least one block stacked properly.

For the time trial mode, it was very similar to the normal mode, but now, we needed to keep track of the time from the start of the game to the moment that the user either wins or loses. To do this, we imported a timer function that would capture the time at the start of the round, and then capture it again once the round has been completed. The timer function doesn't operate like a traditional stopwatch would, instead it captures the current world time; thus, the program would need to subtract the previous time from the current time and then display the difference in seconds during the message displayed when you win or lose.

Lastly, for the endless mode, there were a few more steps that needed to be integrated. First, we needed to add a counter that would increment every time the row was incremented in the loop, this essentially would be the score that is given to the user. At the end of the game, it would display the number of rows that were successfully stacked on top of each other. Next, we needed to include a feature that would keep the game going infinitely as long as there was at least one piece of the current block on top of the stack. To do this, upon reaching the top row, we shifted the entire row down by one every time the middle button was clicked and incremented the speed too. The final step involved adding lighting to the background that would help convey the visual that the stack was moving upward. This involved changing the background colour of every second row that would move down with the stacked block, which gives the perception that they are moving upward.

B. Issues Encountered

During the development phase of the project, we encountered a few issues and bugs. The biggest issue we faced was the implementation of the endless mode. Since we initially started by creating the normal mode followed by the time trial mode, the foundation we made worked for making it to the top and not going further. For the endless mode, we did have to rethink a few functions and their structure as well as add a spot to shift the array down. Overall, this was the issue that took us the longest to overcome. In addition, the initial stages of the program were troublesome too. We started simply by moving just one single pixel back and forth across the screen, but we did not use arrays. So when we tried to translate that to 3 consecutive pixels we had to rethink and recreate the function used to translate these pixels across the screen. A lot of

the issues that we encountered were due to how we coded the program. We created a list of steps to implement and one by one coded them into the program and by trial and error we were able to make them work together.

VI. Project Code

Appendix A contains the file for the main method of the project. In this method, we initialize all variables and start our game loop. A welcome screen is created to allow the user to select and start any mode. In Appendix B, you will see our functions that are used in the various modes, we choose to have this file for the common methods to reduce the length of the project code. We have a function called moveBlock which performs as explained in the following pseudocode:

```
if location is left wall:
     change direction
     move the block by 1 to the right
else if location is right wall:
     change direction
     move the block by 1 to the left
else:
     move block by 1 in the current direction
```

Appendix C, D, and E are the Normal, Time Trial, and Endless Modes respectively. In each mode, there is a game loop and inside the game loop is where we perform the vast majority of algorithms that operate our code. In each loop iteration, we check for button presses, and depending on if a button is pressed or not, we will check the current block position with the stacked tower below and compare to see if blocks need to be removed from the current block as well as clearing the screen of any hanging blocks over the edge of the stacked tower. In the middle of the game loop, if a button is pressed and the user is at the top row, the game will verify if it is a win or loss and return back to the main method allowing the user to select a new mode to go into. The following pseudocode is how our game loop for our normal mode was designed:

```
while True:
     check row to get speed value
     sleep() program for the speed specified

     if button press = true:
          Decrement the current row variable

     Loop through stacked array:
          Check if there any blocks that are
          'floating' to the side of the tower and
          remove them

     If the current row is the 4th or 6th row:
```

```
            Remove one piece from the current block

        If game is over:
            Print message with either win or lose
            Break game loop

        Move the block either to the right or left by 1

        Update the current row colours, change the
        previous location to a clear colour and the new
        location to the colour of the block

        Update the previous line by changing the colour
        of the removed blocks on the screen

End of loop
```

The code for the time trial mode and endless mode has a very similar operation except with the time trial mode we start a timer right before the game loop begins and right after the game has been verified to be over, stops the timer. In the endless mode, we now have a variable that tracks the number of rows cleared and once the user is that the top row, where we check for 'floating' blocks we also now shift the stacked array down by 1 as seen below in the pseudocode:

```
Stacked[7] = stacked[6]
Stacked[6] = stacked[5]
Stacked[5] = stacked[4]
Stacked[4] = stacked[3]
Stacked[3] = stacked[2]
Stacked[2] = stacked[1]
Stacked[1] = current block position
```

In addition, we implemented a loop to go through the rows and change the colour of the background every other row. Every time the block is stacked these coloured rows shift down by one to show the user that they are making progress in stacking the tower.

VII. Project Analysis

A. Analysis

Our project progression went very smoothly throughout the course of the term. We each worked on certain aspects of the code and utilizing a Github repository to merge our copies of the code proved to be vital in limiting the number of errors and bugs. There was no specific leader for the whole project, we each took turns dividing the term into 3 parts and each part had a person who was responsible for organizing zoom meetings and assigning tasks. Looking back at the function of our program there is little that we would change in regards to the look and feel of it but we did find that certain aspects of the code could have been more streamlined allowing for shorter functions. If we had more time we would have been able to go back to try and rewrite the code, possibly add more functions for repeated pieces and even improve the searching through arrays to speed up the loop iterations.

B. Self-Learning Points

Throughout the development of this program, we learnt a good amount of Python and its syntax. In addition, we learned how to emulate a Raspberry Pi and Sense HAT through Oracle VM VirtualBox. We developed critical thinking skills in regards to how we designed each iteration of our game loops. These skills will prove to be important in our careers as software engineers. In addition, working on a Github interface was new for some of us and since Github is a valuable tool used in the industry this was one of the more important pieces. Group projects are also abundant in the field of software engineering, so team building, communication, and time management skills are something that we can carry forward.

C. Conclusion

In conclusion, the development of our program was a success and we were able to produce a Raspberry Pi version of Stacker using the Sense HAT emulator. We incorporated additional features that were unique to our version of the game and it is fully functional. Throughout the development, we have greatly expanded our knowledge of Python together with

how certain data types work and their applications. Due to the COVID-19 pandemic, there were certain features that we would have liked to incorporate that we were unable to. Such features include auditory signals, cleaner and smoother transitions, and a leaderboard system. For the auditory signals, we would have liked there to have been an intro song at the start of the program as well as an audio cue whenever a block was stacked like it is in the arcade game. When running our program, we noticed that the movement of the block felt robotic and not as smooth transitioning as we would have wanted. If given more time, we would like to improve on this and make it a more cleaner and fluid movement. Lastly, we would have wanted to include a leaderboard system that would keep track of the standings and score, but with the emulated Raspberry Pi and Sense HAT, it is much harder to do. In terms of assembly language, we were unable to incorporate it as the Sense HAT emulator and the Python coding language do not intertwine seamlessly with the assembly language in our use case.

Appendix A

```
#
# Stacker.py
#
# Stacker game for the senseHat raspberry PI
#
# Nicholas Imperius
# Jimmy Tsang
# Kristopher Poulin
#

from sense_emu import SenseHat, ACTION_PRESSED
from array import *

import time

from NormalMode import normalMode
from TimeTrialMode import timeTrialMode
from EndlessMode import endlessMode

# CONSTANTS
SPEED = 0.2
CLEAR = (0,0,0)
YELLOW = (244, 208, 63)
PINK = (142, 68, 173)

#
# --Main Function--
#
def main():
    # Declare the hat variable that will be used to control the sense hat
      emulator
    hat = SenseHat()

    # Clear the Screen
    hat.clear()

    # Default layout of the start screen, with 1's representing the blocks to
      colour
    startScreen = [[0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,0,1,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,1,1,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,1,1,1,0,0,0,0],
                   [0,0,0,0,0,0,0,0]]

    # Starting position of selector
    cursorSpot = 2

    # Game Loop to run until program is quit
    checker = 0
    while True:
```

```
# Sleep function allows the cursor to blink
time.sleep(SPEED)

# Print out the start screen layout
for i in range(len(startScreen)):
    for j in range(len(startScreen[i])):
        if startScreen[i][j] == 1:
            hat.set_pixel(j, i, YELLOW)

# Allows us to show either a pink or clear dot for the selector
checker +=1
if checker % 2:
    hat.set_pixel(5, cursorSpot, PINK)
else:
    hat.set_pixel(5, 2, CLEAR)
    hat.set_pixel(5, 4, CLEAR)
    hat.set_pixel(5, 6, CLEAR)

# event checker, when joystick event is pressed
#  it will check the location of the cursor and
#  do the appropriate functions
events = hat.stick.get_events()
for event in events:
    if event.action == ACTION_PRESSED:
        # Default starting location
        currentRow = 7
        currentX = [0,0,1,1,1,0,0,0]

        # Default moving direction
        direction = "left"

        # Default stacked array to represent the
        stacked = [[0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0],
                   [0,0,0,0,0,0,0,0]]

        # Depending on the direction change where the cursor goes
        if event.direction == "up":
            if cursorSpot == 2:
                cursorSpot = 6
            elif cursorSpot == 4:
                cursorSpot = 2
            else:
                cursorSpot = 4
        elif event.direction == "down":
            if cursorSpot == 2:
                cursorSpot = 4
            elif cursorSpot == 4:
                cursorSpot = 6
            else:
                cursorSpot = 2
        elif event.direction == "middle":
```

```
                    # If we are on the top position, show message and start
                      Normal Mode
                    if cursorSpot == 2:
                        hat.clear()
                        hat.show_message("Normal Mode...GO!",
                                                 scroll_speed=0.06)
                        normalMode(currentRow, currentX, direction, stacked,
                                                                      hat)
                        hat.clear()
                    # If we are on the middle position, show message and start
                      Time Trial Mode
                    elif cursorSpot == 4:
                        hat.clear()
                        hat.show_message("Time Trial Mode...GO!",
                                                 scroll_speed=0.06)
                        timeTrialMode(currentRow, currentX, direction,
                                                       stacked, hat)
                        hat.clear()
                    # If we are on the bottom position, show message and start
                      Endless Mode
                    elif cursorSpot == 6:
                        hat.clear()
                        hat.show_message("Endless Mode...GO!",
                                                 scroll_speed=0.06)
                        endlessMode(currentRow, currentX, direction, stacked,
                                                                      hat)
                        hat.clear()
# End of main

# Run Main function
if __name__ == "__main__":
    main()
```

Appendix B

```
#
# stackerUpdater.py
#
# Provides functions that update variables/screen
#
# Nicholas Imperius
# Jimmy Tsang
# Kristopher Poulin
#

# CONSTANTS
CLEAR = (0,0,0)
CYAN = (129, 240, 255)
LIGHT_CYAN = (37, 62, 65)

#
# --MoveBlock Function--
#   - moves the block one pixel to left or right and returns the new direction
#
def moveBlock(direction, currentX):
    # Check if direction is currently right or left
    if direction == "right":
        # Get the position of the block and either move it once more in the
        #  same direction or turn it around and move the other way
        if currentX[7] == 1:
            direction = "left"
            for i in range(0, (len(currentX)), 1):
                if currentX[i] == 1:
                    currentX[i-1] = 1
                    currentX[i] = 0
        elif currentX[0] == 1:
            direction = "right"
            for i in range((len(currentX)-1), 0, -1):
                if currentX[i] == 1:
                    currentX[i+1] = 1
                    currentX[i] = 0
        else:
            for i in range((len(currentX)-1), 0, -1):
                if currentX[i] == 1:
                    currentX[i+1] = 1
                    currentX[i] = 0
    # Check if direction is currently right or left
    elif direction == "left":
        # Get the position of the block and either move it once more in the
        #  same direction or turn it around and move the other way
        if currentX[7] == 1:
            direction = "left"
            for i in range(0, (len(currentX)), 1):
                if currentX[i] == 1:
                    currentX[i-1] = 1
                    currentX[i] = 0
        elif currentX[0] == 1:
            direction = "right"
```

```
            for i in range((len(currentX)-1), -1, -1):
                if currentX[i] == 1:
                    currentX[i+1] = 1
                    currentX[i] = 0
        else:
            for i in range(0, (len(currentX)), 1):
                if currentX[i] == 1:
                    currentX[i-1] = 1
                    currentX[i] = 0

    # Return the direction in case it has been changed
    return direction
# end of moveBlock

#
# --updatePrevLine Function--
#   - clears the pixel that needs to be removed
#
def updatePrevLine(stacked, currentRow, hat):
    index = 0
    # Increments through one row that needs to be looked at
    for x in stacked[currentRow+1]:
        if x == 0:
            hat.set_pixel(index, currentRow+1, CLEAR)
        index += 1
# end of updatePrevLine

#
# --lowerStack Function--
#   - Clears any pixels that arent set when the board is shift down
#
def lowerStack(stacked, hat, index):
    rowIndex = index
    # Go through the entire stackedArray and check to see what needs to be
      changed
    for i in range(1, len(stacked)):
        for j in range(0, len(stacked[i])):
            if stacked[i][j] == 0:
                if rowIndex % 2 == 0:
                    hat.set_pixel(j, i, LIGHT_CYAN)
                else:
                    hat.set_pixel(j, i, CLEAR)
        rowIndex += 1
#end of lowerStack
```

Appendix C

```
#
# normalMode.py
#
# Normal Mode functionality
#
# Nicholas Imperius
# Jimmy Tsang
# Kristopher Poulin
#

from sense_emu import SenseHat, ACTION_PRESSED

import time
from datetime import timedelta

from StackerUpdater import moveBlock, updatePrevLine

# CONSTANTS
SPEED = 0.2
CLEAR = (0,0,0)
CYAN = (129, 240, 255)
YELLOW = (244, 208, 63)
RED = (255, 124, 126)

#
# --normalMode Function--
#
def normalMode(currentRow, currentX, direction, stacked, hat):
    # Variable to keep track of gameLoop
    gameLoop = True

    # Start the game loop, loop exits when user either wins or loses
    while gameLoop:
        # Increase Difficulty as you get Higher
        if currentRow > 4:
            time.sleep(SPEED)
        elif currentRow > 1:
            time.sleep(SPEED * 0.7)
        else:
            time.sleep(SPEED * 0.5)

        # Reset clicked flag
        clicked = False

        # Register click events and increment row system.
        events = hat.stick.get_events()
        for event in events:
            if event.action == ACTION_PRESSED:
                clicked = True
                if currentRow == 0:
                    gameLoop = False
                else:
                    currentRow -= 1
```

```
# Updates array to remove the blocks that are not on top of each other
#  Since this happens after the click we need to search the row that
#  is 2 rows below since we have moved up by one. We compare the
   current
#  row and that row that was 2 below to see if they are the same or
#  different, if different we update the array to remove it when we
#  update the screen later on in this game loop
if (currentRow+1) < 7 and clicked == True:
    for i in range(len(stacked[currentRow+2])):
        if currentX[i] != stacked[currentRow+2][i]:
            currentX[i] = 0
            stacked[currentRow+1][i] = 0

# Removes a piece of the block at the 4th row and the 6th row
if currentRow == 1 and currentX.count(1) != 0:
    numOfOnes = 0
    index = 0
    for i in range(len(currentX)):
        if currentX[i] == 1:
            numOfOnes += 1
            index = i
    if numOfOnes > 1:
        currentX[index] = 0
elif currentRow == 4 and currentX.count(1) != 0:
    numOfOnes = 0
    index = 0
    for i in range(len(currentX)):
        if currentX[i] == 1:
            numOfOnes += 1
            index = i
    if numOfOnes > 2:
        currentX[index] = 0

# Checks if game is done and shows the appropriate message
if currentX.count(1) == 0:
    time.sleep(0.5)
    hat.show_message("You lost.", scroll_speed = 0.05)
    break
elif gameLoop == False and currentX.count(1) != 0:
    time.sleep(0.5)
    hat.show_message("You won!", scroll_speed = 0.05)
    break
elif gameLoop == False and currentX.count(1) == 0:
    time.sleep(0.5)
    hat.show_message("You lost.", scroll_speed = 0.05)
    break


# Call the moveBlock function in StackerUpdater.py to move the block
  and return a new direction
direction = moveBlock(direction, currentX)

# Adjust the screen to colour or uncolour and pixels that need to be
  set or unset
index = 0
for x in currentX:
```

```
            if x == 0:
                hat.set_pixel(index, currentRow, CLEAR)
                stacked[currentRow][index] = 0
            elif x == 1:
                # Checks what row we are at to change to the right colour
                if currentRow > 4:
                    hat.set_pixel(index, currentRow, CYAN)
                    stacked[currentRow][index] = 1
                elif currentRow > 1:
                    hat.set_pixel(index, currentRow, YELLOW)
                    stacked[currentRow][index] = 1
                else:
                    hat.set_pixel(index, currentRow, RED)
                    stacked[currentRow][index] = 1
            index += 1

        # This is where we remove the unwanted blocks from the screen.
        # Calls the updatePrevLine function in StackerUpdater.py
        if clicked == True:
            updatePrevLine(stacked, currentRow, hat)
# end of normalMode
```

Appendix D

```
#
# timeTrialMode.py
#
# Time Trial Mode functionality
#
# Nicholas Imperius
# Jimmy Tsang
# Kristopher Poulin
#

from sense_emu import SenseHat, ACTION_PRESSED

import time
from datetime import timedelta

from StackerUpdater import moveBlock, updatePrevLine

# CONSTANTS
SPEED = 0.2
CLEAR = (0,0,0)
CYAN = (129, 240, 255)
YELLOW = (244, 208, 63)
RED = (255, 124, 126)

#
# --timeTrialMode Function--
#
def timeTrialMode(currentRow, currentX, direction, stacked, hat):
    # Start the timer
    startTime = time.monotonic()

    # Variable to keep track of gameLoop
    gameLoop = True

    # Start the game loop, loop exits when user either wins or loses
    while gameLoop:
        #Increase Difficulty as you get Higher
        if currentRow > 5:
            time.sleep(SPEED * 0.95)
        elif currentRow > 2:
            time.sleep(SPEED * 0.7)
        else:
            time.sleep(SPEED * 0.5)

        # Reset clicked flag
        clicked = False

        # Register click events and increment row system.
        events = hat.stick.get_events()
        for event in events:
            if event.action == ACTION_PRESSED:
                clicked = True
                if currentRow == 0:
```

```
            gameLoop = False
        else:
            currentRow -= 1

# Updates array to remove the blocks that are not on top of each other
#  Since this happens after the click we need to search the row that
#  is 2 rows below since we have moved up by one. We compare the
   current
#  row and that row that was 2 below to see if they are the same or
#  different, if different we update the array to remove it when we
#  update the screen later on in this game loop
if (currentRow+1) < 7 and clicked == True:
    for i in range(len(stacked[currentRow+2])):
        if currentX[i] != stacked[currentRow+2][i]:
            currentX[i] = 0
            stacked[currentRow+1][i] = 0

# Checks if game is done
#  If it is, stop timer, get difference in time and then display a Win
   or Lose with the time it took
if currentX.count(1) == 0:
    endTime = time.monotonic()
    totalTime = timedelta(seconds=endTime - startTime)
    time.sleep(0.5)
    hat.show_message("You lost. Time: " +
 str(totalTime.total_seconds())[:4] + " sec", scroll_speed = 0.05)
    break
elif gameLoop == False and currentX.count(1) != 0:
    endTime = time.monotonic()
    totalTime = timedelta(seconds=endTime - startTime)
    time.sleep(0.5)
    hat.show_message("You won! Time: " +
 str(totalTime.total_seconds())[:4] + " sec", scroll_speed = 0.05)
    break
elif gameLoop == False and currentX.count(1) == 0:
    endTime = time.monotonic()
    totalTime = timedelta(seconds=endTime - startTime)
    time.sleep(0.5)
    hat.show_message("You lost. Time: " +
 str(totalTime.total_seconds())[:4] + " sec", scroll_speed = 0.05)
    break


# Call the moveBlock function in StackerUpdater.py to move the block
  and return a new direction
direction = moveBlock(direction, currentX)

# Adjust the screen to colour or uncolour and pixels that need to be
  set or unset
index = 0
for x in currentX:
    if x == 0:
        hat.set_pixel(index, currentRow, CLEAR)
        stacked[currentRow][index] = 0
    elif x == 1:
        if currentRow > 5:
            hat.set_pixel(index, currentRow, CYAN)
```

```
                    stacked[currentRow][index] = 1
                elif currentRow > 2:
                    hat.set_pixel(index, currentRow, YELLOW)
                    stacked[currentRow][index] = 1
                else:
                    hat.set_pixel(index, currentRow, RED)
                    stacked[currentRow][index] = 1
            index += 1

        # This is where we remove the unwanted blocks from the screen.
        # Calls the updatePrevLine function in StackerUpdater.py
        if clicked == True:
            updatePrevLine(stacked, currentRow, hat)
# end of timeTrialMode
```

Appendix E

```
#
# endlessMode.py
#
# Endless Mode functionality
#
# Nicholas Imperius
# Jimmy Tsang
# Kristopher Poulin
#

from sense_emu import ACTION_PRESSED

import time

from StackerUpdater import moveBlock, updatePrevLine, lowerStack

# CONSTANTS
CLEAR = (0,0,0)
CYAN = (129, 240, 255)
LIGHT_CYAN = (27, 70, 75)
LIGHT_YELLOW = (71, 75, 27)
LIGHT_RED = (75, 27, 28)

#
# --endlessMode Function--
#
def endlessMode(currentRow, currentX, direction, stacked, hat):
    # Initialize variables to be used in the Game Loop
    gameLoop = True
    speedFlag = True
    speed = 0.2

    # Keeps track of rows completed
    numberOfCompletedRows = 0

    # Start the game loop, loop exits when user either wins or loses
    while gameLoop:
        # Increase Difficulty as you get Higher
        if numberOfCompletedRows > 10 and speedFlag == True:
            speed -= 0.001
        elif currentRow == 5 and speedFlag == True:
            speed = speed
        elif currentRow == 3 and speedFlag == True:
            speed = speed * 0.8
        elif currentRow == 1 and speedFlag == True:
            speed = speed * (0.6 / 0.8)

        # Sleep for the amount of time, controls speed of the stacker block
        time.sleep(speed)

        # speedFlag is used when the button has been pressed to increment the
        #  speed, every loop we set to false before the event loop
        speedFlag = False
```

```
# Reset clicked flag
clicked = False

# Register click events and increment row system.
events = hat.stick.get_events()
for event in events:
    if event.action == ACTION_PRESSED:
        speedFlag = True
        numberOfCompletedRows += 1
        clicked = True
        # Once we get to the top row, we can no longer decrement
        if currentRow != 0:
            currentRow -= 1

# Updates array to remove the blocks that are not on top of each other
#  Since this happens after the click we need to search the row that
#  is 2 rows below since we have moved up by one. We compare the
   current
#  row and that row that was 2 below to see if they are the same or
#  different, if different we update the array to remove it when we
#  update the screen later on in this game loop
if (currentRow+1) < 7 and clicked == True:
    if numberOfCompletedRows < 8:
        for i in range(len(stacked[currentRow+2])):
            if currentX[i] != stacked[currentRow+2][i]:
                currentX[i] = 0
                stacked[currentRow+1][i] = 0
    # If we are at the top row, we need to shift the entire array down
    #  by one so we can show that they are moving "up"
    else:
        fakeCurrentX = [0,0,0,0,0,0,0,0]
        for i in range(len(stacked[currentRow])):
            if currentX[i] != stacked[1][i]:
                currentX[i] = 0
                fakeCurrentX[i] = 0
            else:
                fakeCurrentX[i] = stacked[0][i]
        stacked[7] = stacked[6]
        stacked[6] = stacked[5]
        stacked[5] = stacked[4]
        stacked[4] = stacked[3]
        stacked[3] = stacked[2]
        stacked[2] = stacked[1]
        stacked[1] = fakeCurrentX

# Checks if game is done, this happens when currentX is full of 0's
if currentX.count(1) == 0:
    # Need to decrement rows since we increment at the start of the
      loop
    numberOfCompletedRows -= 1
    # Print message to user displaying the number of rows completed
    hat.show_message("Your Score: " + str(numberOfCompletedRows) + "
                                          rows.", scroll_speed = 0.05)
    # break out of game loop
    break
```

```
        # Call the moveBlock function in StackerUpdater.py to move the block
          and return a new direction
        direction = moveBlock(direction, currentX)

        # Adjust current moving row of pixels
        # Will print a different colour block depending on what row they are
          on
        index = 0
        for x in currentX:
            if x == 0:
                if numberOfCompletedRows > 6:
                    if numberOfCompletedRows % 3 == 0:
                        stacked[currentRow][index] = 0
                        if numberOfCompletedRows % 2 == 1:
                            hat.set_pixel(index, currentRow, LIGHT_CYAN)
                        else:
                            hat.set_pixel(index, currentRow, CLEAR)
                    elif numberOfCompletedRows % 3 == 1:
                        stacked[currentRow][index] = 0
                        if numberOfCompletedRows % 2 == 1:
                            hat.set_pixel(index, currentRow, LIGHT_CYAN)
                        else:
                            hat.set_pixel(index, currentRow, CLEAR)
                    elif numberOfCompletedRows % 3 == 2:
                        stacked[currentRow][index] = 0
                        if numberOfCompletedRows % 2 == 1:
                            hat.set_pixel(index, currentRow, LIGHT_CYAN)
                        else:
                            hat.set_pixel(index, currentRow, CLEAR)
                else:
                    hat.set_pixel(index, currentRow, CLEAR)
                    stacked[currentRow][index] = 0
            elif x == 1:
                hat.set_pixel(index, currentRow, CYAN)
                stacked[currentRow][index] = 1
            index += 1

        # This is where we remove the unwanted blocks from the screen.
        # Calls the updatePrevLine function in StackerUpdater.py
        if clicked == True and numberOfCompletedRows < 7:
            updatePrevLine(stacked, currentRow, hat)
        elif clicked == True and numberOfCompletedRows > 6:
            if numberOfCompletedRows % 3 == 0:
                lowerStack(stacked, hat, numberOfCompletedRows)
            elif numberOfCompletedRows % 3 == 1:
                lowerStack(stacked, hat, numberOfCompletedRows)
            elif numberOfCompletedRows % 3 == 2:
                lowerStack(stacked, hat, numberOfCompletedRows)
# end of endlessMode
```