Create Performance Task

Row 4 - Procedural Abstraction

Instructions

1. You can only use this version if you can write on a pdf, otherwise please use the ppt version.
2. Read the criteria for this row on slides 3 - 4.
3. For each response *(slides 5 – 14)*:
   a. Underline any code, phrases or sentences that meet any of the criteria.
   b. Tick (✓) any criteria that are satisfied and cross (✗) any criteria that are not satisfied.
   c. Write a **1** in the bottom right corner if the response gets the mark for this row *(all criteria have been satisfied)*, otherwise write a **0**.
4. When finished, save this presentation as a pdf with 2 slides to a page and submit this file.

Scoring Criteria

- The written response:
  - includes two program code segments:
    - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
    - one showing where the student-developed procedure is being **called**.
  - describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

Decision Rules

- Consider ONLY written response 3c when scoring this point.
  - Requirements for program code segments:
    - The procedure must be student developed, but could be developed collaboratively with a partner.
    - If multiple procedures are included, use the first procedure to determine whether the point is earned.
  - Do NOT award a point if any one or more of the following is true:
    - The code segment consisting of the procedure is not included in the written responses section.
    - The procedure is a built-in or existing procedure or language structure, such as an event handler or main method, where the student only implements the body of the procedure rather than defining the name, return type (if applicable) and parameters.
    - The written response describes what the procedure does independently without relating it to the overall function of the program.

Important Note

- On the slides in this presentation, I have often cropped screenshots so that I can easily fit everything on one slide
- However, the first screenshot should show ALL the code in the function and second should show at least some of the surrounding code.

```
int check_overall(char current_grid[6][7], char player)
{
    for (int i = 0; i < 6; i ++)
    {
        for (int j = 0; j < 7; j ++)
        {
            if (current_grid[i][j] == player)
            {
                if (check_individual(i, j, current_grid) == 1)
                {
                    return 1;
                }
            }
        }
    }
    return 2;
}
```

a

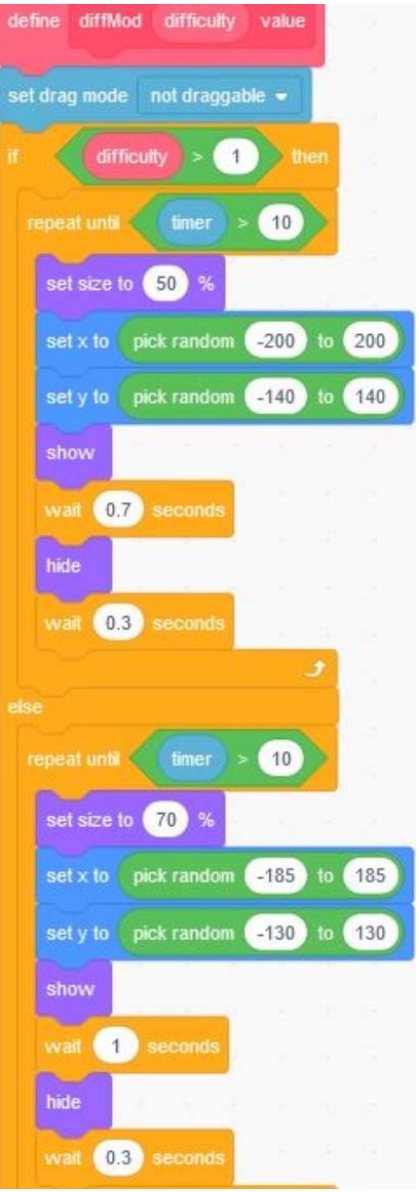
```
if (check_overall(grid, current_turn) == 1)
{
    turn = 0;
}
else
{
    turn ++;
}
```

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

3.c.iv: The function (check_overall) is given an argument "current_grid" which is the updated array "grid" after each players turn and the argument "player" which is the player whose turn it was. It then works by using a "for loop" that runs six times, then nesting another "for loop" inside that runs seven times. The outer loop will initialize the variable "i" as 0 and increment by +1 each time it runs, while the inner loop will initialize the variable "j" as 0 and increment by +1 each time it runs. In the inner loop, If the piece belongs to "player", check_overall will call the check_individual function and pass in "i," "j," and "current_grid" as arguments. This function will access the item current_grid[i][j] and will determine whether a vertical, horizontal, or diagonal sequence of four pieces of the same owner stems from that position. If it does determine such a sequence, check_individual will return 1 and save it into the "status" variable. Otherwise, check_individual will return 2 and save into "status." As check_overall iterates through all forty-two positions, if "status" is ever 1, the function will return 1 and exit the check_overall function. If "status" never equals 1 after iterating through every position, check_overall will return 2.

3.c.iii: After each player's turn, this function iterates through each item in the array (each position on the game board) to determine if a row of four like-pieces stems from it. This contributes to the overall functionality of the program by checking after every player's turn to see if they've won.

?



b

3.c.iii:

The identified procedure above accounts for the difficulty modifier that is accomplished by a parameter that accepts values based on a ask/answer block where one target can change in size or the wait time with specific boundaries to prevent a sprite to be at the edge. This program contributes to the overall functionality of the program because without the program the output:score and output: targetsHit cannot be updated because the two inputs mouse click and difficulty cannot be established.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?

```
def move_character(event, name, speed)
    global final_score
    name_box = Text(name)
    name_box.set_position(3,15)
    name_box.set_font("10pt Arial")
    add(name_box)
    if event.key == "ArrowLeft":
        character.move((speed*-1), 0)
```

C

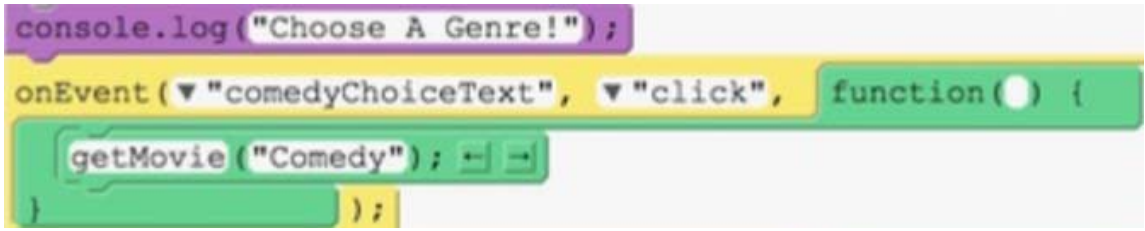
```
def call_function(event):
    if (character.get_x()!=400) and (character.get_y()!=415):
        if speed_input == "2x":
            move_character(event, "Player: " + player, 5)
        elif speed_input == "3x":
            move_character(event, "Player: " + player, 15)
    else:
        results_page()
add_key_down_handler(call_function)
```

3.c.iv: It moves the red ball depending on which mouse key the user inputted through the use of selection. If the user pushes the right key the ball moves to the right a certain amount, if user pushes the left key it moves to the left, if user enters the up key the ball moves up, and if user pushes the down key it moves down. After each time the ball moves, the procedure uses a for loop and if statements to check if the ball is on a coordinate of the walls, or the score items such as the stars, by accessing the lists that these coordinates are located in and by checking if even one of those coordinates matches the balls current coordinates. If the coordinates of the ball matches a coordinate of the wall, the procedure lets the user know that they bumped into a wall and makes them restart with zero points, however if the ball's coordinates match one of the coordinates of the score items, then it lets the user know that they got some points. It keeps track of these points by appending a number to a list each time the user gets a point. It then multiplies the length of the list by the amount of points of each item, and stores that in a variable which is the user's final score.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?

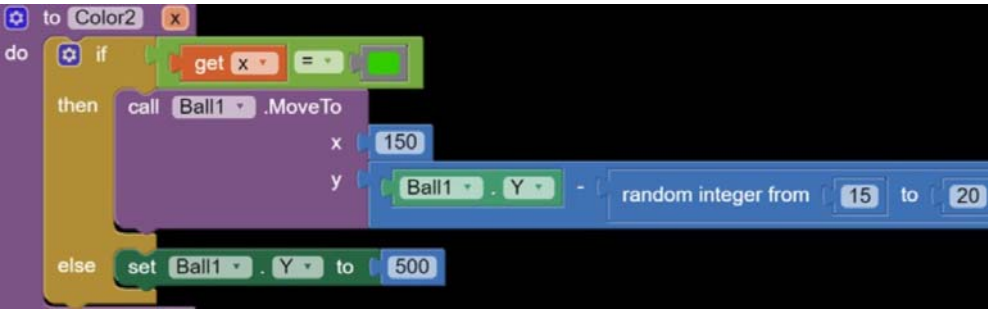
d



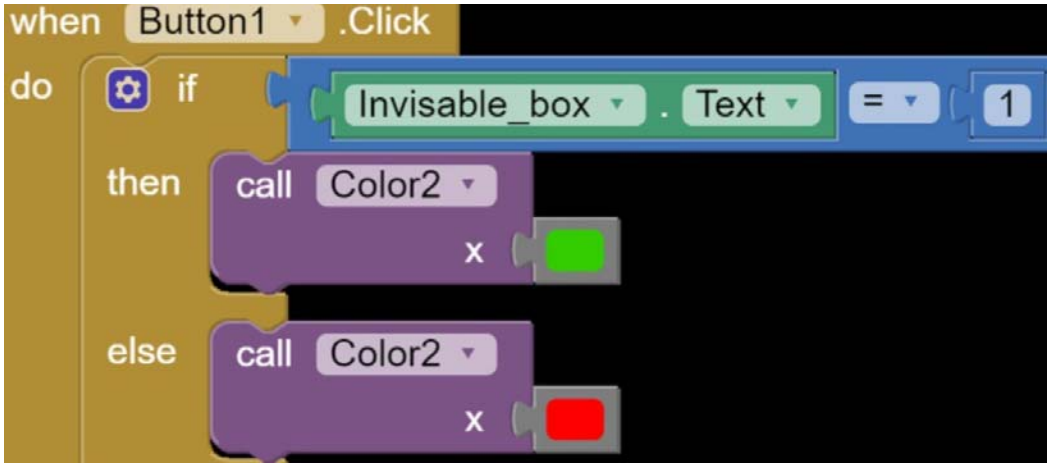
3.c.iii. In the function pictured above, the code is using the input of the user's age and genre choice to from there, randomly select what three movies the user will be suggested. Without this function in the program, the user's genre and age would not be implemented into the overall decision of what list to pull movies from. But with it, the user is able to receive age-appropriate and correct genre movie suggestions.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?



e



3.c.iii.

The identified procedure decides whether the ball moves forward or if it gets sent back to the beginning based on what color the screen is.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?

```
function findTotal(hours) {
  var total = 0;
  for (var i = 0; i < hours.length; i++) {
    total = total + hours[i];
    setText("totaloutput", "You've listened to " + (total + " hours of music this week"));
    if (total > 27) {
      setText("iftotal>27", "Wow! You've already listened to more music than the average person per
week! ");
    }
  }
}
```

f

```
onEvent("addButton", "click", function( ) {
  appendItem(hours, getNumber("inputHours"));
  findTotal(hours);
```

3.c.iii.

The function is necessary in order to calculate the total number of hours of music the user has listened to and tell the user they've listened to more music than the average person per week if their total is greater than 27.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?

```
function filter(userChoice) {
  var filteredList = [];
  var output = "";
  for (var i = 0; i < quoteList.length; i++) {
    if (quoteType[i] == userChoice) {
      appendItem(filteredList, quoteList[i]);
      output = output + quoteList[i] + "\n";
    }
  }
  setText("outputText", output);
}
```

g

```
onEvent("movieButton", "click", function( ) {
  type = "movie line";
  filter(type);
});
onEvent("songButton", "click", function( ) {
  type = "song lyric";
  filter(type);
});
```

3.c.iii.

The filter function contributes to the overall functionality by filtering one-by-one through my list in order to output a filtered list that matches the user input.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?

```
function housevssenate(input) {
  var stateNames = getColumn("Fema
  var senateTotals = getColumn("Fe
  var houseTotals = getColumn("Fer
  var cumulativeSenate = 0;
  var cumulativeHouse = 0;
  for (var i = 0; i < stateNames.
    if (stateNames[i] == input) {
```

h

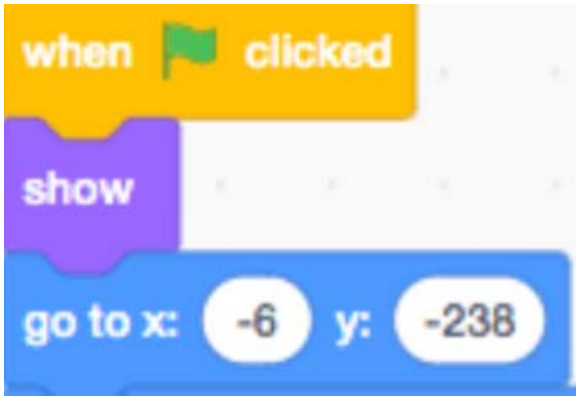
```
onEvent("pickastate", "change", function() {
  var input = getText("pickastate");
  housevssenate(input);
});
```

3.c.iii.

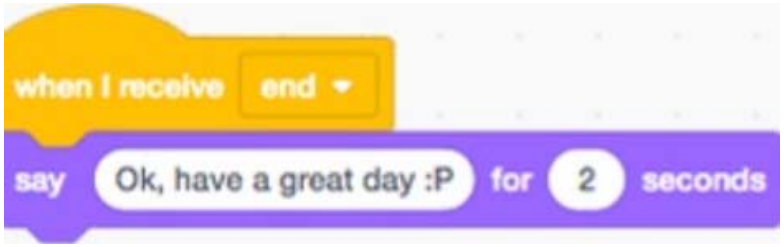
The selected procedure states whether the House or the Senate has more female representatives from 1981 to 2019 in the list. It takes into account the state the user selects in order to give the result.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?



i



3.c.iii.

The procedure identified is the decision of the user to participate in the trivia game or not. This was important to implement into the code because I wanted the user to have the option even though it is more likely than not that the user will want to participate in the game.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?

j

```
def Mainprogram ():
 strikes = 0
 points = 0
 repeat1 = 0
 repeat2 = 0
 repeat3 = 0
 repeat4 = 0
 repeat5 = 0
 print( "Wecome To f
chose a scenery. All
```

```
elif user == 5:
  print("The villian is
  user2 = input("Where
  if repeat5 == 1:
    print("hamcking you
  elif user2 in listE:
```

3.c.iii.

The procedure is what the program need to be able to use the options as well to see how many point will give you a wining ending or a losing ending. This procedure contributes to the function of the program by being the backbone of the entire program and holding everything.

- includes two program code segments:
  - one showing a student-developed **procedure** with at least one **parameter** that has an **effect** on the functionality of the procedure.
  - one showing where the student-developed procedure is being **called**.
- describes what the identified procedure **does** and how it **contributes to the overall functionality** of the program.

?