Create – Applications From Ideas
Written Response Submission Template

Please see Assessment Overview and Performance Task Directions for Student for the task directions and recommended word counts.

Program Purpose and Development

2a)

My program was created using Processing which uses the Java program language. The program I created is a game named "Wall Ball." The purpose of the game is to let the player control the paddle at the bottom of the screen with their mouse. The player must use the paddle to hit the moving puck which will increase their score. The game ends if the player misses the puck and it touches the bottom of the screen. The video demonstrates how the game is played by showing off how to move the paddle and how to hit the puck. The video also shows the game over screen that the player will be taken to if they lose. This screen displays the players final score as well as the high score. The video then shows that the user can click the buttons to play again.

2b)

When writing my program, I independently broke down my code down into several small functions in order to better organize my code and make it easier to read. I then organized these functions into different sections depending on which screen I wanted them to run on.

One difficulty I faced was the task of creating the different screens. I needed to figure out how to run only the home screen functions on the home screen, the game screen functions on the game screen, and so on. To resolve this issue, I independently created a string variable named "screen" and passed it into a parameter of a function named displayScreen(). This function used the value of "screen" to decide which screen the program should display.

Another difficulty I faced when writing my program was figuring out how to draw the tiny dots that block the users vision when they play the game. To fix this issue, I independently made several arrays that stored the information needed to draw these dots. I then looped through the arrays to initialize all of the values in them. With all of that data, I was able to easily draw the dots to the screen.

2c)

```
443
444
445
446
447 //changes the speed of the puck when it hits a wall
448 void updatePuckSpeed()
449 {
450   if (puckX > width - PUCK_SIZE || puckX < 0)
451   {
452     puckSpeedX *= -1;
453   }
454   if (puckY < BAR_Y + BAR_HEIGHT)
455   {
456     puckSpeedY *= -1;
457   }
458 }
459
460 //makes the puck bounce off of the paddle
461 void bouncePuckOffPaddle()
462 {
463   if (puckY+PUCK_SIZE>blockY&&puckY<blockY+BLOCK_HEIGHT&&puckX+PUCK_SIZE>blockX&&puckX<blockX+BLOCK_WIDTH)
464   {
465     puckSpeedY *= -1;
466     puckSpeedY--;
467     score++;
468     for (int i = 0; i < NUM_DOTS; i++)
469     {
470     dotsSize[i]++;
471     }
472   }
473 }
474
475 //moves the puck based upon the values of puckSpeedX and puckSpeedY
476 void movePuck()
477 {
478   puckX += puckSpeedX;
479   puckY += puckSpeedY;
480 }
481
482
483
```

The algorithm that I independently wrote is movePuck() which is created by using the two sub-algorithms updatePuckSpeed() and bouncePuckOffPaddle().

The purpose of bouncePuckOffPaddle() is to make the puck change direction whenever it touches the paddle. It does this by checking the values of puckX and puckY to see if they have crossed over with the coordinates of the paddle. If they have, then the value of puckSpeedY will be updated.

The purpose of updatePuckSpeed() is to make the puck change directions whenever it hits the edges of the screen. This is done by checking the values of puckX and puckY to see if they are out of the range of the screen. If they have, either puckSpeedX or puckSpeedY will be updated.

movePuck() combines both of these algorithms by using the variables puckSpeedY and puckSpeedX that both of the sub-algorithms are constantly updating. It adds these values to puckX and puckY which helps to fulfill the purpose of the game by making the puck respond to events such as touching the paddle or hitting an edge. Without this algorithm, the puck would only move in a single direction and eventually go off the screen.

2d)

```
159  /*This abstraction allows me to efficiently create new
160  buttons and choose what screen they take you to when they are clicked.*/
161  void button(int x, int y, int bWidth, int bHeight, String pickScreen)
162  {
163    if (mouseX > x && mouseX < x + bWidth && mouseY > y && mouseY < y + bHeight)
164    {
165      fill(125);
166      if (mousePressed && timer == 0)
167      {
168      screen = pickScreen;
169      if(pickScreen == "gameScreen")
170      {
171          reset();
172      }
173      timer = 30;
174      }
175    }
176    else
177    {
178      if (timer > 0)
179      {
180      timer--;
181      }
182      fill(0);
183    }
184    rect(x, y, bWidth, bHeight);
185  }
```

The abstraction I chose is a function I independently wrote named button(). When this function is called, it draws a button that the user can click on to travel between the different screens in my program. This function helps to manage the complexity of my program due to the fact that if I didn't have it, I would have to write out the same lengthy and confusing code each time I needed a button. Since I have six different buttons in my program, my program would quickly grow in length and be harder to read if I didn't use this abstraction. Parameters also help to manage complexity because they make the function more versatile. When this function is called, values are placed into the parameters that customize each button as necessary. My abstraction uses the logical operators "&&" to state that every condition must be true in order to run the code in the if statements. My abstraction also uses arithmetic expressions to calculate the range of coordinates that are over the button. If the mouse is located in one of these coordinates, the button will change color to show that it can be clicked on.