b 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)

I created my game using ALICE which uses the Java programming language. The purpose of my game is for the wolf to "eat" five blue chickens before the timer runs out. The video depicts all of the chickens first changing color, then the wolf eating the blue chickens when they are clicked on. If the blue chickens run out, all of the chickens will change color again so that the goal of the game can be completed.

2b)

I independently developed this program starting with my original method and continuing to build off of that. During the process, as I would think of an element that I needed to add to the game, I would work on the code needed for that certain part of the game. An element that gave me some trouble was the first part my wolfMove method which was meant to have the wolf move to blue chickens and eat them. I had a few problems with this method because at first, my wolf would move, but end up halfway underground. After a lot of trial and error, I was able to make the wolf move up while moving forward and make sure that when it stopped moving it was above ground. I was also presented with opportunities to add aspects to make my game better. One of these moments was when I decided that my game was too easy to play. To make the game more fun, I looked up a tutorial on how to make a countdown timer. Once I added the timer to the game, there was more pressure, which made the game realistic.

2c)

```
world.wolfMove ( [Obj] chicken)
  No variables
  // moves wolf to blue chickens and "eats" them
  If ( chicken . color == [blue] )
      wolf turn to face chicken
      While ( ( wolf distance to chicken ) > 1 )
          wolf move amount = 1 meter toward target = chicken duration = .08 seconds
          wolf move up 0.1 meters duration = 0 seconds
      wolf move to chicken duration = 0 seconds
      wolf move up ( ( ( subject = wolf 's height ) / 2 ) ) duration = 0.1 seconds
      Do together
          chicken set opacity to 0 (0%)
          chicken set color to [black]
      // the counting method is called here
      world.scoreCounter
      // color reset method below
      world.resetColors
  Else
    Do Nothing
```

```
world.resetColors ( )
 No variables
  // keeps track of the number of blue chickens and resets the colors if there are none
  left
  world.numBlue set value to 0
  For all world.chickens , every item_from_chickens together
    If ( item_from_chickens . color == [blue] )
      world.numBlue set value to ( ( world.numBlue + 1 ) )
    Else
      Do Nothing
  For all world.chickens , every item_from_chickens together
    If ( world.numBlue == 0 )
      world.setrandomcolor chicken = item_from_chickens
    Else
      Do Nothing
```

```
world.scoreCounter ( )
 No variables
  // counts the number of chickens eaten
  increment world.counter by 1
  chickenCounter set text to ( world.counter as a string )
```

I chose to use my “wolfMove” method, which I wrote independently, that contains the code that tells my wolf how to move, as well as two other methods “resetColors” and “scoreCounter”. “wolfMove” uses math to tell the wolf how to move towards the blue chickens and sets their opacity to zero. The method moves the wolf to the blue chickens and then the chickens’ opacity is set to zero. The “resetColors” method uses math to count the number of blue chickens and proceeds to use logic and call my “setrandomcolors” (I will talk more about this method in 2D) method if there are no blue chickens left. My “scoreCounter” method uses math to count the number of chickens that have been “eaten” or disappeared.

2d)

```
world.setrandomcolor ( [Obj] chicken)
  randomNum = 1
    // Gives all the chickens different colors
    randomNum set value to ( random number minimum = 1 maximum = 9 integerOnly =
  true )
    If ( randomNum == 1 )
        chicken set color to [red] duration = 0.5 seconds
    Else
        If ( randomNum == 2 )
            chicken set color to [blue] duration = 0.5 seconds
        Else
            If ( randomNum == 3 )
                chicken set color to [magenta] duration = 0.5 seconds
            Else
                If ( randomNum == 4 )
                    chicken set color to [blue] duration = 0.5 seconds
                Else
                    If ( randomNum == 5 )
                        chicken set color to [red] duration = 0.5 seconds
                    Else
                        If ( randomNum == 6 )
                            chicken set color to [blue] duration = 0.5 seconds
                        Else
                            If ( randomNum == 7 )
                                chicken set color to [magenta] duration = 0.5 seconds
                            Else
                                If ( randomNum == 8 )
                                    chicken set color to [orange] duration = 0.5 seconds
                                Else
                                    If ( randomNum == 9 )
                                        chicken set color to [blue] duration = 0.5 seconds
                                    Else
                                        Do Nothing
```

The abstraction I chose is my method “setrandomcolor.” This method uses math to set a random number to each of the chickens that are used in my game. The method then uses logical if/else statements to assign different colors to the range of numbers that are given to the chickens. This abstraction helped manage the complexity of my program because it ensured that I would not have to put together the large amount of code every time that I called the method. I call this method once in the beginning of my program and then later if there are not any blue chickens left. Because of this abstraction I did not have to re-write the same ten if/else statements more than once.