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 app, Salmon Picker, is written in Javascript. The purpose of the program is to provide entertainment and a fun challenge for users. With your mouse, start by clicking on the salmon scattered throughout the maze. If the mouse touches a maze wall or moves off of the screen, you lose the game. In order to click on the "boxed-in" salmon, you must click on the red buttons attached to the walls. Once clicked, the red buttons will make the white walls disappear. Once the user has successfully clicked on a salmon, the salmon counter at the top of the page will update the existing score by one. To win the level, the user must collect all salmon within the allotted time. If you do not collect all the salmon in the maze before the timer hits zero, you lose the game. Because they have more walls and salmon to collect, harder levels provide a fun task.

2b)

First, I designed the three playing screens. This ensured that each level could be adjusted to varying degrees of difficulty. For harder screens, I programmed the timer to count down from a smaller number, increased the amount of salmon within the maze, or implemented a more intricate maze design. After I rendered each maze entertaining, I programmed all the salmon, maze walls, and salmon counter. One difficulty in my program development arose when the salmon counter would not reset to zero when the user attempted a level again but rather started counting up from a previous score. After independently experimenting with a global variable that initially set the click score to zero, I found that reassigning the value of each click score to zero when the "start over" button was pressed solved the issue. Then, I worked iteratively to adjust timer allowances and the locking of playing levels. Another difficulty in my program development arose when the timer would not run when the moving maze obstacle in level three traveled up and down the screen. To resolve this, I independently changed the setTimeout control's wait-before-execution time to 2000 milliseconds, allowing the obstacle to sync it's movement with the timer's.

2c)

```
//Parent Algorithm created independently
function timer2() {
  timedLoop(1000, function() {
   obMove();
   seconds=seconds-1;
   obMove1();
   setText("labelTime", seconds);
   setText("labelTime2", seconds);
   setText("labelTime3", seconds);
   if (seconds===0) {
     setScreen("lostScreen");
   }
   if (seconds <=0 && click < 9) {
     setScreen("lostScreen");
     stopTimedLoop();
   }
   if (seconds>0 && click===9) {
   setScreen("wonmediumScreen");
   stopTimedLoop();}
   });
}
//Child Algorithm 1 created independently
function obMove() {
  for (var i = 0; i < 20; i++) {
    moveblockOb38();
  }
}
//Child Algorithm 2 created independently
function obMove1() {
  for (var i = 0; i < 20; i++) {
    blockmoveOb38();
  }
}
```

My main algorithm is timer2() which indicates how many seconds have elapsed since starting the level. Additionally, timer2() ends the game if all the salmon has not been collected within the time limit. Timer 2() has two key parts: obMove() and obMove1(). obMove() loops the function moveblockOb38() twenty times so that level 3's moving block can travel all the way down the screen. After the value of the moving obstacle's Y position has increased by five, obMove() repeats this action which makes the obstacle move down. The obMove1() function loops the function blockmoveOb38() twenty times so that the moving obstacle in level 3 has the ability travel up a screen after moving to the bottom of the screen. After obMove() has executed, obMove1 brings the moving obstacle back to the top of the page by looping the action of decreasing the moving obstacle's Y position by 5. My main algorithm combines obMove() and obMove1() by enabling the moving block to move only when the timer starts counting down. Together these algorithms help achieve the purpose of my program by challenging the user to click on the salmon in the path of the moving block while remaining untouched by it.

2d)

```
//Abstraction which I created independently
function timer() {
  timedLoop(1000, function() {
   seconds=seconds-1;
   setText("labelTime", seconds);
   setText("labelTime2", seconds);
   setText("labelTime3", seconds);
   if (seconds===0) {
     setScreen("lostScreen");
   }
   if (seconds <=0 && click < 7) {
     setScreen("lostScreen");
     stopTimedLoop();
   }
   if (seconds>0 && click===7) {
   setScreen("woneasyScreen");
  stopTimedLoop();}
   });
}
```

My abstraction, function timer(), counts down from the number assigned to the global variable, "seconds," every 1000 milliseconds. Timer() manages the complexity of my program by enabling me to call it in multiple areas of my program without having to rewrite the code for it each time. Also instead of implementing and running multiple subtraction operators every time 1000 milliseconds have elapsed, the timer function uses a timedLoop that updates how much time has elapsed by creating one single subtraction operator that subtracts the number one from "seconds." If this function did not exist, I would have to rewrite the timer()'s code for each of the three levels, creating unnecessary lines of code. Additionally, I would have to keep reassigning a number to "seconds" every 1000 milliseconds if this function did not exist.