Create PT - Written Response Template

Assessment Overview and Performance Task Directions for Students

Video Submit one video in .mp4, .wmv, .avi, or .mov format that demonstrates the running of at least one significant feature of your program. Your video must not exceed 1 minute in length and must not exceed 30MB in size

Prompt 2a. Provide a written response or audio narration in your video that:
- identifies the programming language;
- identifies the purpose of your program; and
- explains what the video illustrates.

(Must not exceed 150 words)

My program is a game made in Javascript. The purpose of the game is to create something fun and enjoyable for the person playing the game by having the player move a ship and dodge enemy ships while collecting food to increase the score. The video shows the ship being moved and dodging the enemies and then being hit and dying. It then shows resetting the game to play again and using the invulnerability power up to not take damage for a couple of seconds.

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and / or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development. *(Must not exceed 200 words)*

First I decided on JavaScript in the App Lab because it is easy to add a UI in it's design mode. I then brainstormed ideas for what type of game to make and eventually decided on a 2d game where you control a ship and dodge enemies. I first developed a collision system with a timed loop. I decided to store the moves inside of a 2d array. One opportunity I saw was when I realized that instead of having to make every single move by hand and not knowing where the player was in real time, I could use an algorithm that checked how far away the player was and then generated a move that moved it as close as possible to the player. I encountered a difficulty when I needed a way to read each move and then move the enemies over time to that location. I wanted to use a for loop but there was no way to slow down the enemies and I couldn't use a timed loop inside the for loop. After trial and error, I used a timed loop to scan each move and generate a new one when the first was done.

2c. Capture and paste a program code segment that implements an algorithm (marked with an **oval** in **section 3**) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, and must integrate mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. *(Must not exceed 200 words)*

Code Segment

```
//parent algorithm
function hit_timer_start() {
  timedLoop(17, function() {
    if (power_true_false == false) {
      setProperty("player","background-color",rgb(26,188,156));
      if
(collision_detector(e1coordsx,e1coordsy,ewidth,eheight,pcoordsx,pcoordsy,pwidth,pheight)) {
        kill();
        console.log("hit");
        stopTimedLoop();
      }
      if
(collision_detector(e2coordsx,e2coordsy,ewidth,eheight,pcoordsx,pcoordsy,pwidth,pheight)) {
        kill();
        console.log("hit");
        stopTimedLoop();
      }
    }else{setProperty("player","background-color","yellow")}
    //collision detector
    if (collision_detector(foodx,foody,foodwh,foodwh,pcoordsx,pcoordsy,pwidth,pheight)){
      ate_food();
    }
    //set positions; "draw"
    setPosition("food",foodx,foody);
    setPosition("enemy1",e1coordsx,e1coordsy);
    setPosition("enemy2",e2coordsx,e2coordsy);
    setText("score_label","Score: "+score);
    setText("time_label",time+" :Time");
  });
}
});
//child algorithm 1
function collision_detector(x1,y1,width1,height1,x2,y2,width2,height2) {
  var true_false = false;
  if ((x1+width1>x2)&&(x1<x2+width2)&&(y1<y2+height2)&&(y1+height1>y2)) {
    true_false = true;
  }
  return true_false;
}
//child algorithm 2
function kill() {
  setScreen("dead_screen");
  setText("dead_time","Your time was: "+time);
  setProperty("dead_time","text","Your time was: "+time+" seconds.");
  setProperty("dead_score","text","Your score was: "+score);
  stop_timer = true;
}
```

Written Response

The algorithm I chose is inside the “hit_timer_start” function. The algorithm first starts a loop which contains everything. It first checks whether the invulnerability power up is active. If it checks if the player is touching any of the enemies with my first sub-algorithm “collision_detector”. This algorithm uses parameters and an if-statement to check if something is touching something else. If it is, it sets the variable true_false to true and after the if statement, returns the value of true_false. After my main algorithm uses “collision_detector”, if the player is touching an enemy it uses my second sub-algorithm “kill”. This algorithm sets the screen to “dead_screen” where it sets the text of the labels to the score and time the player achieved while they were playing the game. It then sets the value of variable “stop_timer” to true which stops the time keep going up. My main algorithm then checks if the user is touching the food and after sets the locations of the enemies and the players to where their coordinate variables say they should be. Together these algorithms add a way to kill the player. These help the purpose of the game by making it fun to play.

2d. Capture and paste a program code segment that contains an abstraction you developed individually on your own (marked with a **rectangle** in **section 3**). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. *(Must not exceed 200 words)*

Code Segment

```
function generate_move(speed) {

  var e1x;var e1y;var e2x;var e2y;
  //check enemy1

  if (pcoordsy - e1coordsy >= Math.max(
      e1coordsy-pcoordsy,pcoordsx-e1coordsx,e1coordsx-pcoordsx)) {
    //down
    if (pcoordsy - e1coordsy < 40) {
      speed_mod = 50;
    }else{speed_mod = 0;}
    e1y = pcoordsy;
  }else if (e1coordsy-pcoordsy >= Math.max(
      pcoordsy - e1coordsy,pcoordsx-e1coordsx,e1coordsx-pcoordsx)) {
    //up
    if (pcoordsy - e1coordsy < 40) {
      speed_mod = 50;
    }else{speed_mod = 0;}
    e1y = pcoordsy;
  }else if (pcoordsx-e1coordsx >= Math.max(
      e1coordsy-pcoordsy,pcoordsy - e1coordsy,e1coordsx-pcoordsx)) {
    //right
    if (pcoordsy - e1coordsy < 40) {
      speed_mod = 50;
    }else{speed_mod = 0;}
    e1x = pcoordsx;
  }else if (e1coordsx-pcoordsx >= Math.max(
      e1coordsy-pcoordsy,pcoordsx-e1coordsx,pcoordsy - e1coordsy)) {
    //left
    if (pcoordsy - e1coordsy < 40) {
      speed_mod = 50;
    }else{speed_mod = 0;}
    e1x = pcoordsx;
  }

  //check enemy2

  if (pcoordsy - e2coordsy > Math.random(
      e2coordsy-pcoordsy,pcoordsx-e2coordsx,e2coordsx-pcoordsx)) {
    //down
    if (pcoordsy - e2coordsy < 40) {
      speed_mod = 50;
    }else{speed_mod = 0;}
    e2y = pcoordsy+20;
  }else if (e2coordsy-pcoordsy > Math.random(
      pcoordsy - e2coordsy,pcoordsx-e2coordsx,e2coordsx-pcoordsx)) {
    //up
    if (e2coordsy-pcoordsy < 40) {
      speed_mod = 50;
    }else{speed_mod = 0;}
    e2y = pcoordsy+20;
  }else if (pcoordsx-e2coordsx > Math.random(
      e2coordsy-pcoordsy,pcoordsy - e2coordsy,e2coordsx-pcoordsx)) {
    //right
    if (pcoordsx-e2coordsx < 40) {
      speed_mod = 50;
    }else{speed_mod = 0;}
    e2x = pcoordsx+20;
  }else if (e2coordsx-pcoordsx > Math.random(
      e2coordsy-pcoordsy,pcoordsx-e2coordsx,pcoordsy - e2coordsy)) {
    //left
    if (e2coordsx-pcoordsx < 40) {
      speed_mod = -20;
    }else{speed_mod = 0;}
    e2x = pcoordsx+20;
  }
  add_move(e1x,e1y,e2x,e2y,speed+time+speed_mod);

}
```

Written Response

The abstraction I chose is the “generate_move” function. This function generates an item for the list “moves” by calculating the distance the player is from the enemies. It also generates the speed of how fast the enemies should move. This function is used multiple other times in the “move_enemies” function to make the first move and each subsequent move. Therefore, the abstraction helps manage complexity because of the single function letting the algorithm be used multiple times. If I didn’t use this abstraction, I would have to repeat the code 8 times which would make it more messy and harder to understand.

Export or save this document as a PDF and turn in to the AP Digital Portfolio along with your **Video** and **Program Code** (separate files).