# 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**

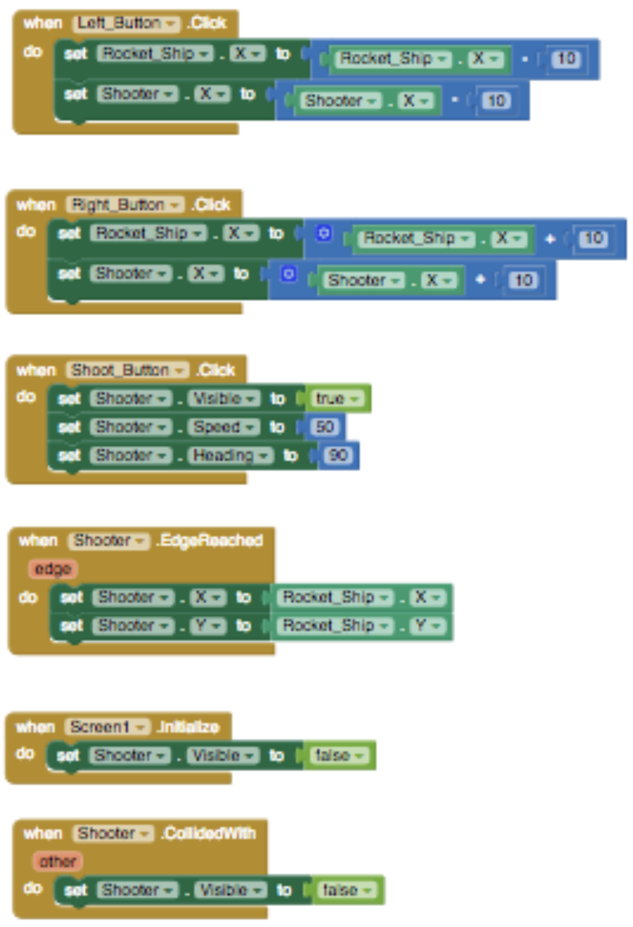
2a)

The purpose of the program is to provide a game for entertainment, it is a game that is based on the old game space invader. I used the programming language in MIT App Inventor. In the video, the gram shows the different aspects of the game. The rocket ship sprite is able to be moved using the right and left buttons. And when the shoot button is clicked, a shooter is shot in the direction and location of where the rocket is currently at. Once you click the start button, you have 15 seconds to shoot all of the asteroids. If you are unable to shoot the asteroids, then the games stops. If the shooter hits the asteroid, then the score changes by 1 point. There is also the option to stop and reset the game which also reset the timer.

2b)

A programming difficulty was the inability to be able to have the shooter shoot from where the rocket is to the asteroid because they are both originally in different canvases. In order to fix this, I had to connect the two canvases and create one canvas. As a result the movement of the asteroids were changed to only being able to move left to right. A big problem with the asteroids were, once the asteroid collided with each other, they would be 'destroyed' but it was only necessary for the asteroid to be 'destroyed' when shot by the shooter. Therefore, it was necessary to simplify the movements, but still made the game difficult by have the asteroids move at different speeds each round. Furthermore, when having the code update the score, the score would change when the code was under the shooter collide function. Therefore I created a separate algorithm to update a score and used it under the asteroid functions.Finally in order to make sure the timer for the game was plausible, I have to try different intervals for the clock, and found the right interval of 15000.

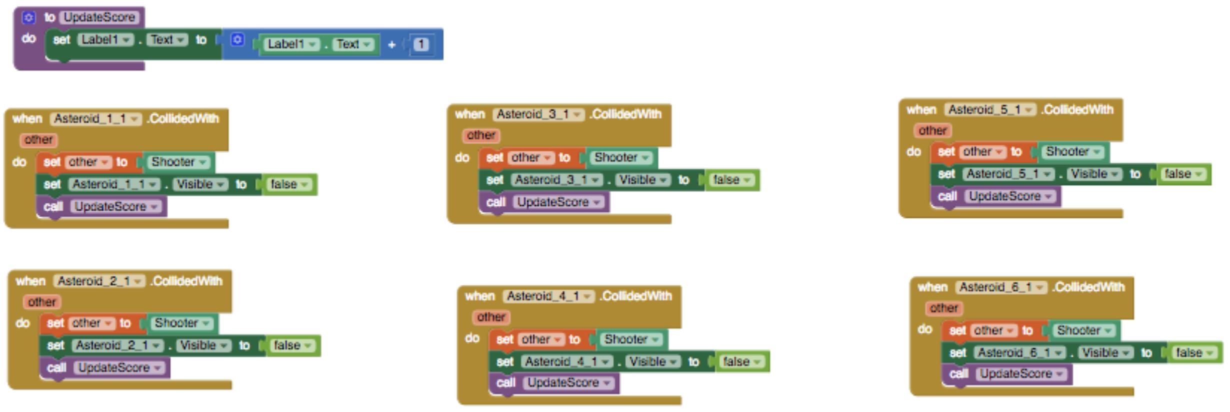
2c)



```
when Left_Button.Click
do  set Rocket_Ship.X to Rocket_Ship.X - 10
    set Shooter.X to Shooter.X - 10

when Right_Button.Click
do  set Rocket_Ship.X to Rocket_Ship.X + 10
    set Shooter.X to Shooter.X + 10

when Shoot_Button.Click
do  set Shooter.Visible to true
    set Shooter.Speed to 50
    set Shooter.Heading to 90

when Shooter.EdgeReached
  edge
do  set Shooter.X to Rocket_Ship.X
    set Shooter.Y to Rocket_Ship.Y

when Screen1.Initialize
do  set Shooter.Visible to false

when Shooter.CollidedWith
  other
do  set Shooter.Visible to false
```

This code controls the movement of the rocket and the shooter in order to play the game. The left, right, and shooter button are depicted in this code. Independently, when the left button is clicked, the rocket has to move to the left, so the position moves -10 spaces. However, because the shooter has to be shot from the rocket, the position of the shooter also moves -10 spaces as well. The right button acts similarly, instead the rocket and shooter move right +10 spaces. The movement to add and subtract 10 from the position of the rocket and shooter is an example of a mathematical concept. Also, the initialization of the screen and the movement of the shooter, function independently from the movement of the rocket. The shooter is connected to the movement of the rocket, so the code is together, and the initialization of the screen begins with the shooter starting invisible. Because this program has specific buttons, it is hard to further abstract the code and create a larger algorithm that would be used to describe the movement of the rocket. However, together these smaller algorithms work together to achieve and create a bigger parent algorithm.

2d)



```
to UpdateScore
do  set Label1 . Text to  Label1 . Text + 1

when Asteroid_1_1 .CollidedWith
  other
do  set other to Shooter
    set Asteroid_1_1 . Visible to false
    call UpdateScore

when Asteroid_3_1 .CollidedWith
  other
do  set other to Shooter
    set Asteroid_3_1 . Visible to false
    call UpdateScore

when Asteroid_5_1 .CollidedWith
  other
do  set other to Shooter
    set Asteroid_5_1 . Visible to false
    call UpdateScore

when Asteroid_2_1 .CollidedWith
  other
do  set other to Shooter
    set Asteroid_2_1 . Visible to false
    call UpdateScore

when Asteroid_4_1 .CollidedWith
  other
do  set other to Shooter
    set Asteroid_4_1 . Visible to false
    call UpdateScore

when Asteroid_6_1 .CollidedWith
  other
do  set other to Shooter
    set Asteroid_6_1 . Visible to false
    call UpdateScore
```

As mentioned before, MIT App Inventor does not accomplish the same level of abstraction that other programs could accomplish, but still manages to have abstraction to manage the complexity of the program. This includes the abstraction to update the score as the game is being played. First I created the function UpdateScore which also uses the mathematical concept of adding 1 to the current number each time the score is updated. Then I called onto that function on each of my asteroid blocks in order to update the score when the asteroids were hit. This way, I used generalization as a form of abstraction to update the score of the game, and manage the complexity of the program. This abstraction is also important for each function of the asteroids colliding with each other, so the code would not have had to been repeated. In another program, instead of the specific variables for each asteroid, it could be a variable for each type of asteroid, and the code would have been much more generalized.