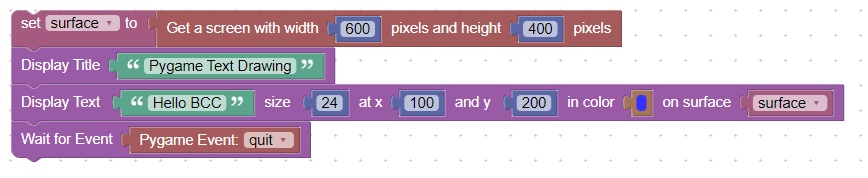
2D Game Design with Blockly Pygame

Paul Richards 9/23/2020Description

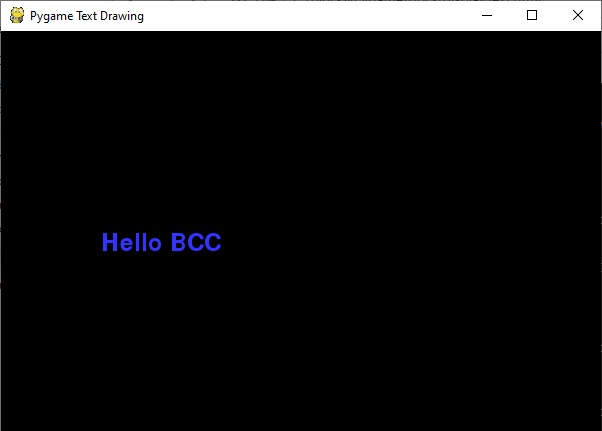
This course is an introduction to 2D Game programming course in python, using blockly-pygame. Blockly-pygame is a visual **programming language** that allows students to create their own interactive stories, games and animations. As students design **Blockly-pyGame** projects, they learn to think creatively, reason systematically, and gain solid test-debugging skills.

No previous programming skills are required, however setup requires the installation of python3, git for windows and pygame.

Students assemble "puzzle-pieces" in logically order under supervision and direction from the instructor. Here is what an example code looks like that outputs "Hello BCC" in the pyGame window:



When this code is executed, it appears in the pyGame application window:



This course also teaches basic programming constructs such as: variables, if statements, boolean logic, functions, function calls, and event-driven-programming as well as pygame specific features for drawing text, lines, circles, images, and animation.

We start with a "Hello World" app, and then progress to a storybook app, followed by tic-tac-toe, and then finish with a fully-functional asteroids-like app complete with sound.

The course is divided into 3 sessions. Each session consists of 8, 1 hour classes. The first sessions deals with setup, basic drag-and-drop blockly-pyGame actions, saving a project, and how to build python code for test and debugging.

Session II discusses utility functions that will be used for the final game. One such utility function is movePlayer which moves the player image based on the pressing of wasd keys, which most students are familiar with.

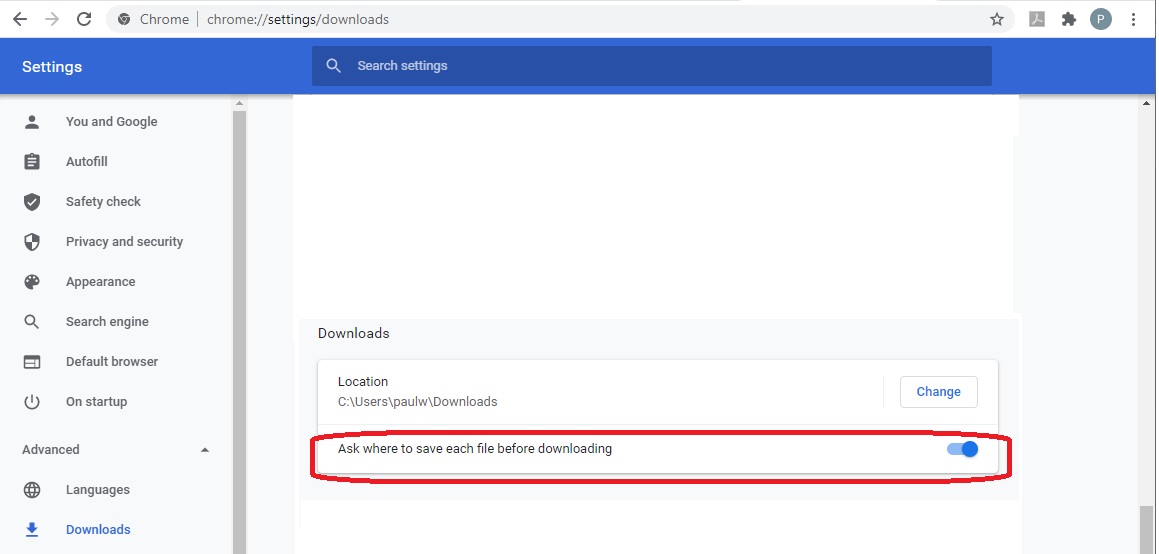
Session III deals with the assembly and testing of the asteroids-like game: Blasteroids.

Detailed Syllabus

1. Session 1: Setup and Pygame Events
   1. Setup, Hello World draw text
   2. Mouse Pressed Event, draw an image
   3. Mouse Released Event, draw lines, and circles
   4. Determine Mouse Position, Functions
   5. Boolean Variables, Finish 1st game: Tic-Tac-Toe
   6. Mouse Movement Event, Track Mouse with Circle
   7. Drag and Drop Event, Checkers boards
   8. Time Event, capture in Checkers
2. Utilities and game concepts
   1. Rotate an image to point at the mouse
   2. Functions that return values
   3. "Fire" a cannonball on key up
   4. Detect mouse click on image
   5. Ok Message Box
   6. Ok-Cancel Message Box
   7. Get key from the user
   8. Input Box
3. Game Creation
   1. Move image based on wasd key
   2. Point image at mouse
   3. Play sound
   4. Drag and Drop (Continued)
   5. Sprite creation
   6. Sprite collision detection
   7. Call an external function
   8. Blasteroids final game assembly

1.1 Setup and Hello World Draw Text

* Install python3
* Install pygame from [here](https://www.pygame.org/download.shtml)
* Install git from [here](https://git-scm.com/downloads)
* Open a cmd prompt and enter the command: git clone https://www.github.com/Paulware/blocklyPygame
* Google Chrome will need to be setup to allow downloads:(Settings, Advanced)



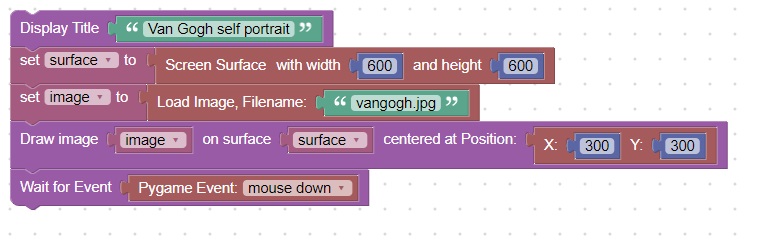
1.2 Mouse Pressed Event, Draw an Image

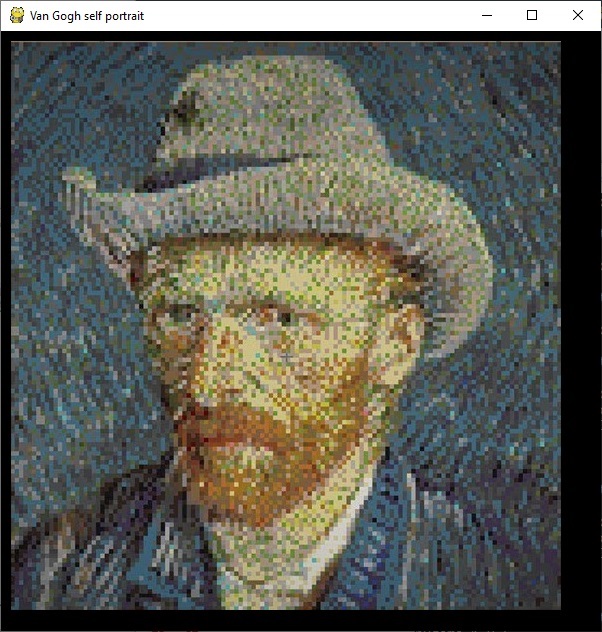
New Concepts

|  |  |
| --- | --- |
| Concept | Description |
| 1 | An Image can be loaded when filename is in the same directory |
| 2 | Y coordinates start at zero at the top and increases as you go down |
| 3 | An image can be displayed with an x,y coordinate on the screen |

Code

This code will draw an image of Van Gogh until a mouse button is pressed.



It will look like:  
  


Display an image and text (storybook)

New Concepts

|  |  |
| --- | --- |
| Concept | Description |
| 1 | Clear the screen after each page |
| 2 | Text and images can be combined one a single page |

Code

This code will create a short story.



1.3 Draw 2 Lines until mouse up, draw circle, and tic-tac-toe board

New Concepts

|  |  |
| --- | --- |
| Concept | Description |
| 1 | Lines can be drawn on the screen using Draw Line block |

Code

This code will draw 2 lines in blue.  
The window will close when the mouse is released



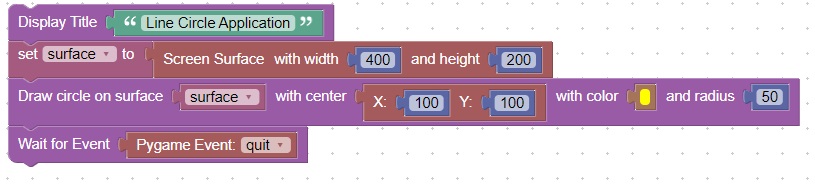
Draw a Circle

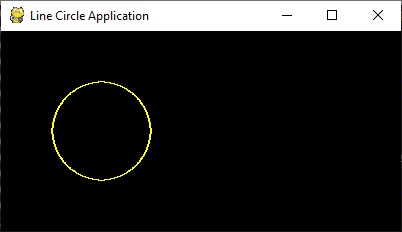
New Concepts

|  |  |
| --- | --- |
| Concept | Description |
| 1 | Circles can be drawn using the draw circle block |

Code

This code will draw a circle in yellow until a mouse button is pressed.

  
Here is what the screen looks like



Draw a tic-tac-toe board

New Concepts

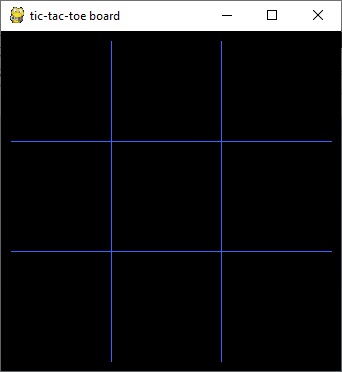
|  |  |
| --- | --- |
| Concept | Description |
| 1 | Circles and lines can be combined on the same screen |

Code

This code will draw a tic-tac-toe board.



Here is what the screen looks like



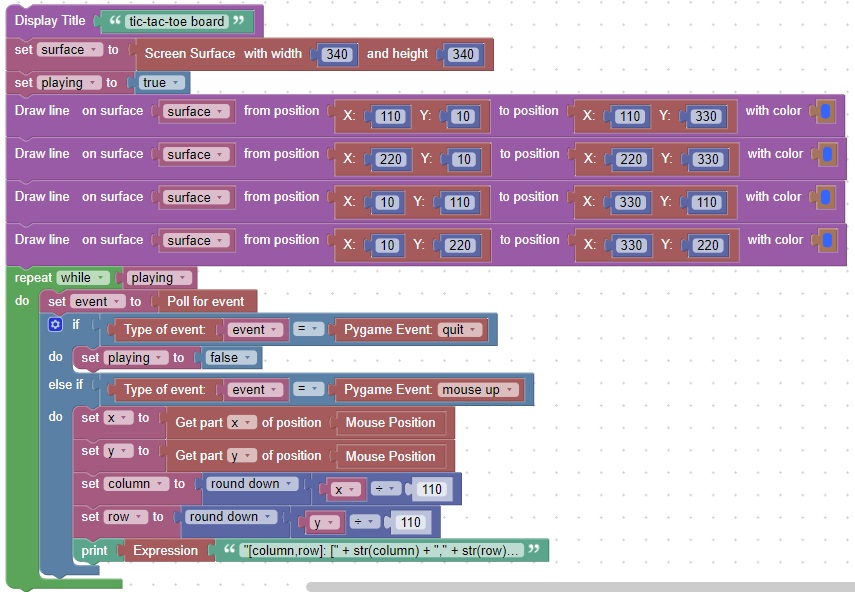
1.4 Convert Mouse Position to player move

New Concepts

|  |  |
| --- | --- |
| Concept | Description |
| 1 | You can create a variable and name it |
| 2 | You can use the set block to set a variable |
| 3 | While Loops |
| 4 | If Statement |
| 5 | Math : Round Down |

Code

This code will determine the row and column that the mouse was clicked on.  
And print the [column,row] value to the console.



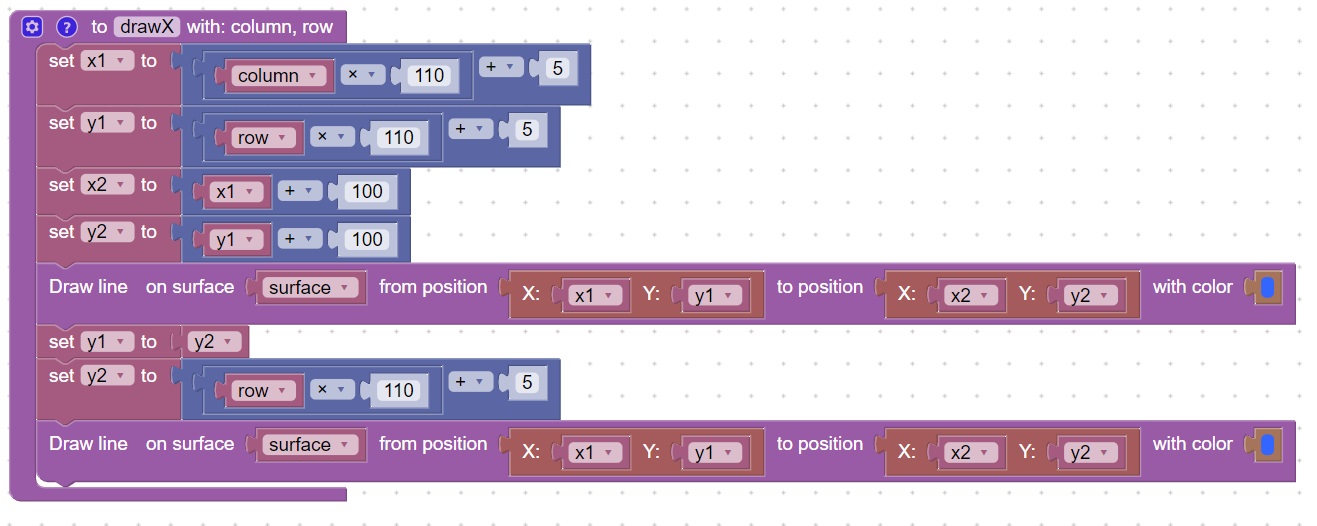
Create a function to draw a tic-tac-toe X

New Concepts

|  |  |
| --- | --- |
| Concept | Description |
| 1 | Functions are groups of code blocks that are all related to a single objective |
| 2 | Parameters are inputs that are passed to the function like (column,row) |
| 3 | Functions can be called from the main function |

Code

This code will create a function that draws an X at a specific column,row.

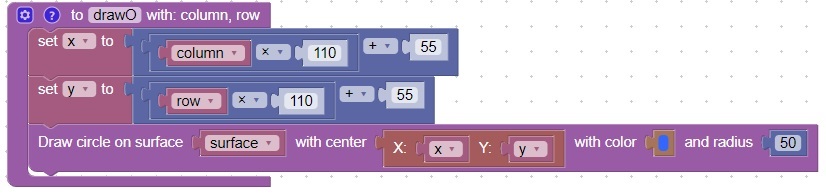


1.5 Draw a tic-tac-toe O  
New Concepts

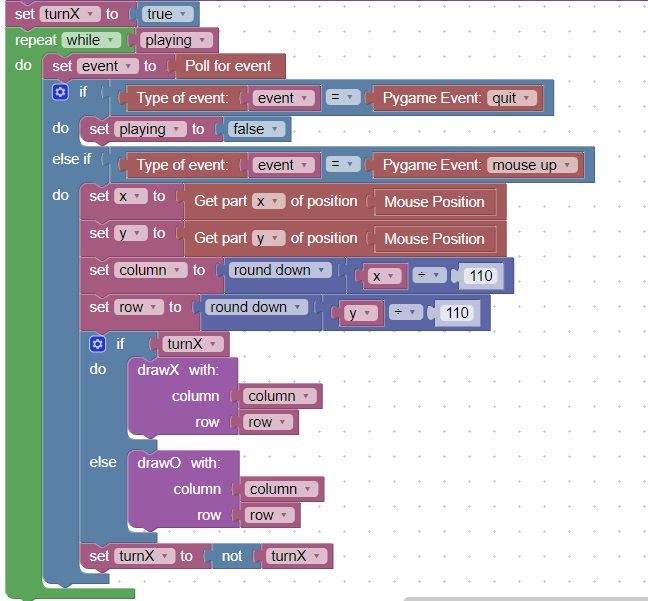
|  |  |
| --- | --- |
| Concept | Description |
| 1 | Boolean variable can be True or False |
| 2 | Boolean Not will change a True to False or change a False to True |
| 3 | If Else allows you to conditionally execute certain code based on variable values |

Code

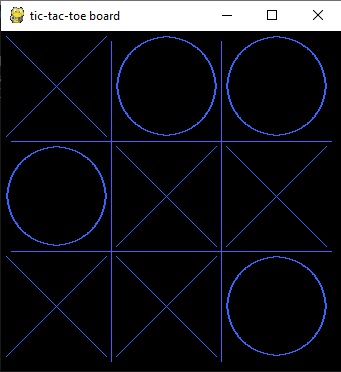
This code will create a function for the tic-tac-toe game that will draw an O



Additional code needs to call either drawX or drawO based on whose turn it is.  
This code will alternate between writing an X and writing an O



This is how the screen looks:

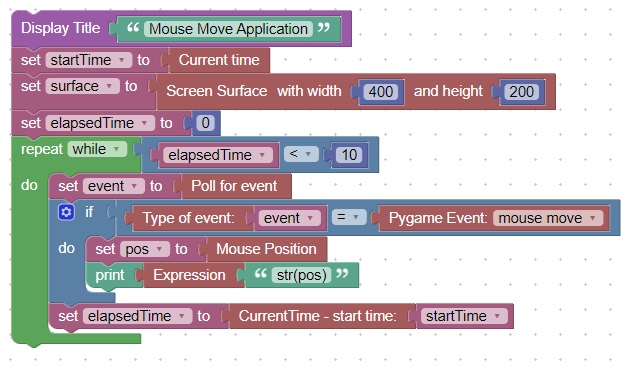


1.6 Mouse Move Event  
New Concepts

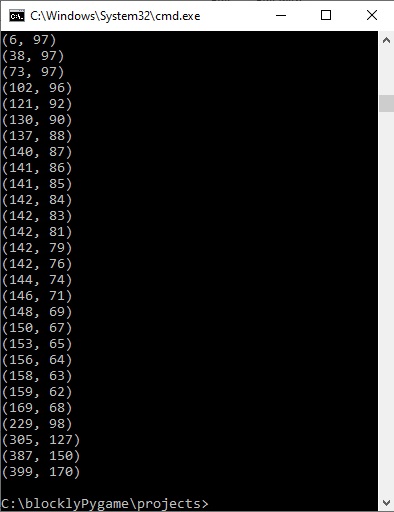
|  |  |
| --- | --- |
| Concept | Description |
| 1 | Mouse position can be determined on mouse move event |

Code

This code will show the mouse position for 10 seconds.



This is what the line terminal looks like

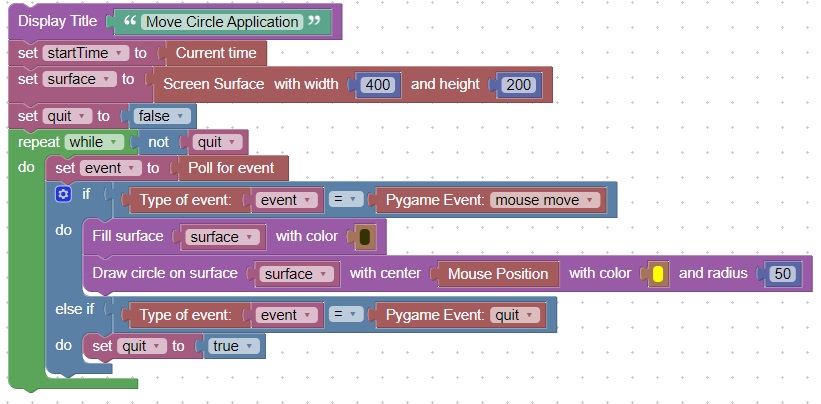


1.7 Move circle wherever the mouse is  
New Concepts

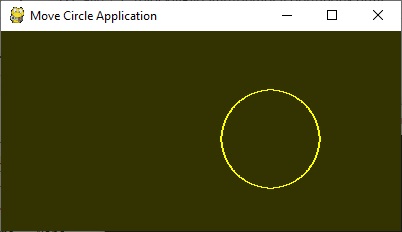
|  |  |
| --- | --- |
| Concept | Description |
| 1 | Circles can be drawn at the position of the mouse when it moves |
| 2 | Screen should be cleared before each draw, to avoid multiple circles appearing |

Code

This code will move a circle until quit event occurs.



Here is what the screen looks like



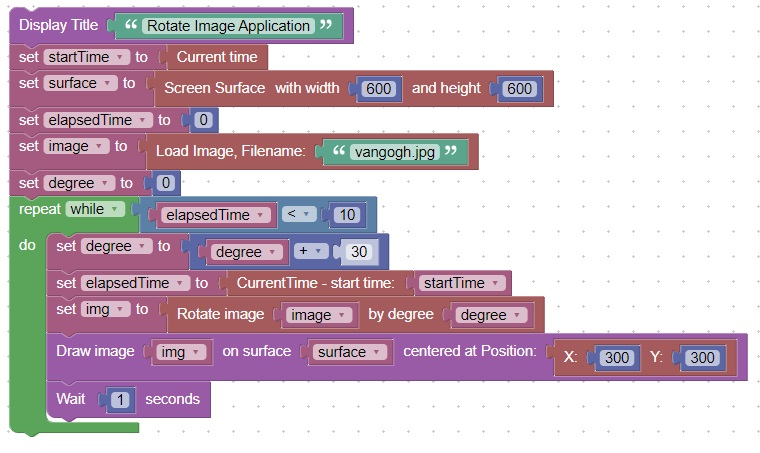
1.8 Images can be rotated  
Rotate Image

New Concepts

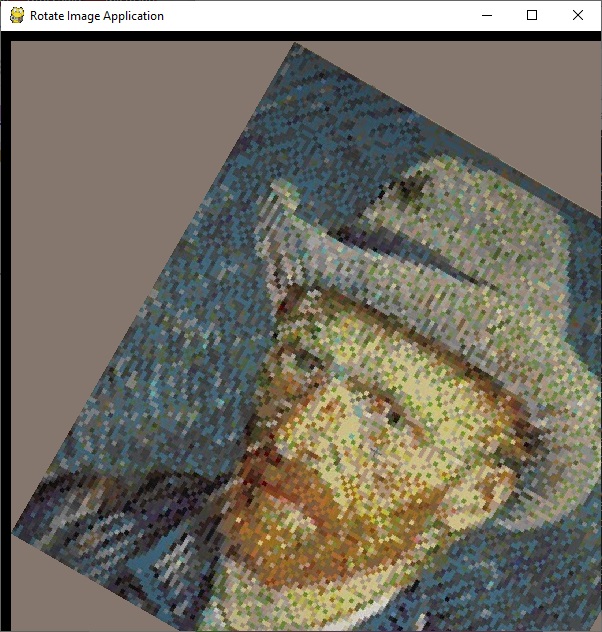
|  |  |
| --- | --- |
| Concept | Description |
| 1 | Images can be rotated using the Rotate image block |

Code

This code will rotate an image for 10 seconds.



Here is what the screen looks like

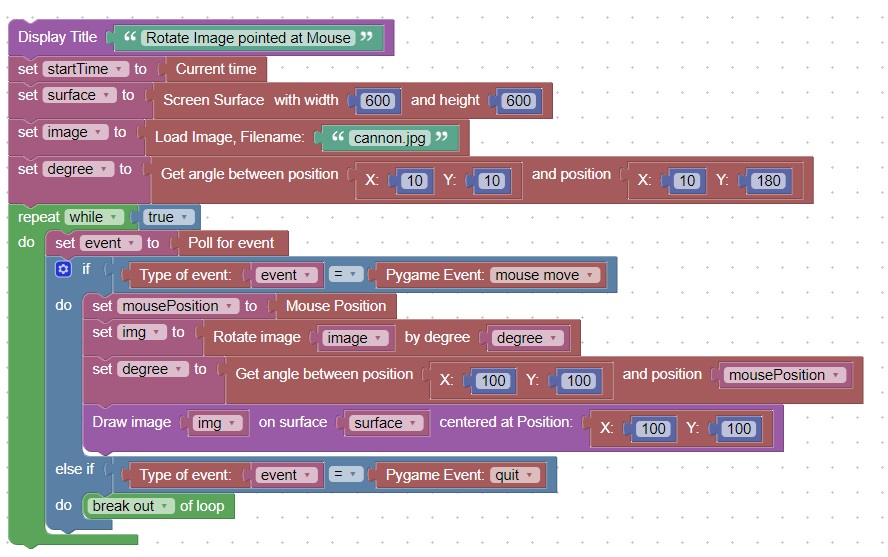


2.1 Rotate an image to point at the mouse  
New Concepts

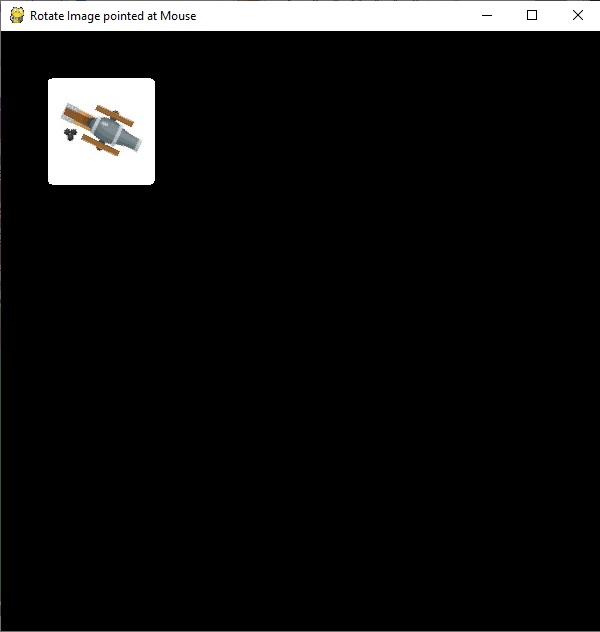
|  |  |
| --- | --- |
| Concept | Description |
| 1 | The angle between 2 positions can be determined using the Get angle block |

Code

This code will rotate an image to point at the mouse until the quit event occurs.



Here is what the screen looks like



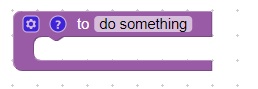
2.2 Function Types

New Concepts

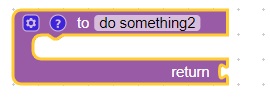
|  |  |
| --- | --- |
| Concept | Description |
| 1 | Void and Return functions are possible with or without parameters |

Code

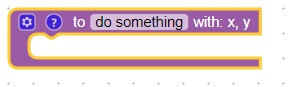
This code will a create void function



This code will create a return function

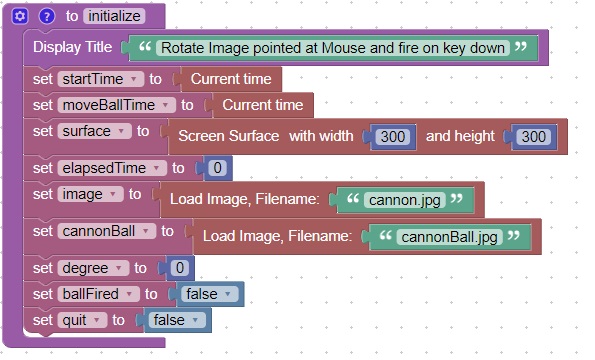


To add a parameter, click on the settings wheel

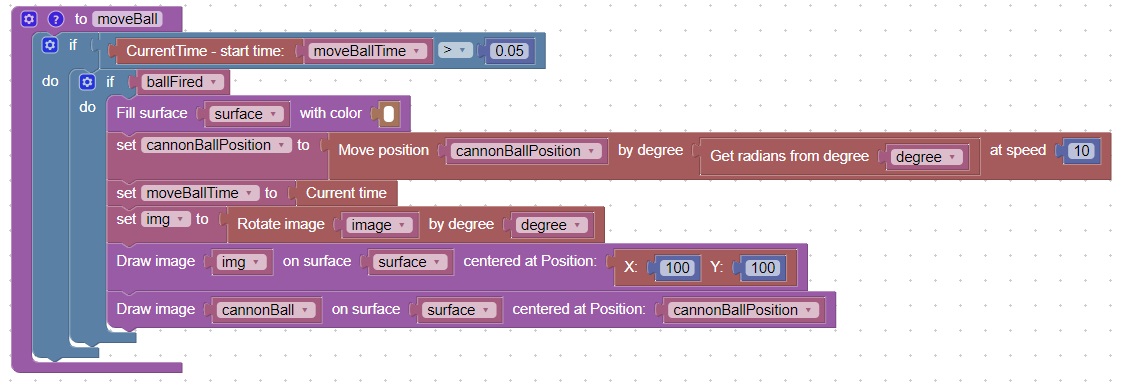


2.3 Rotate an Image to point at the mouse, then fire when a key is pressed

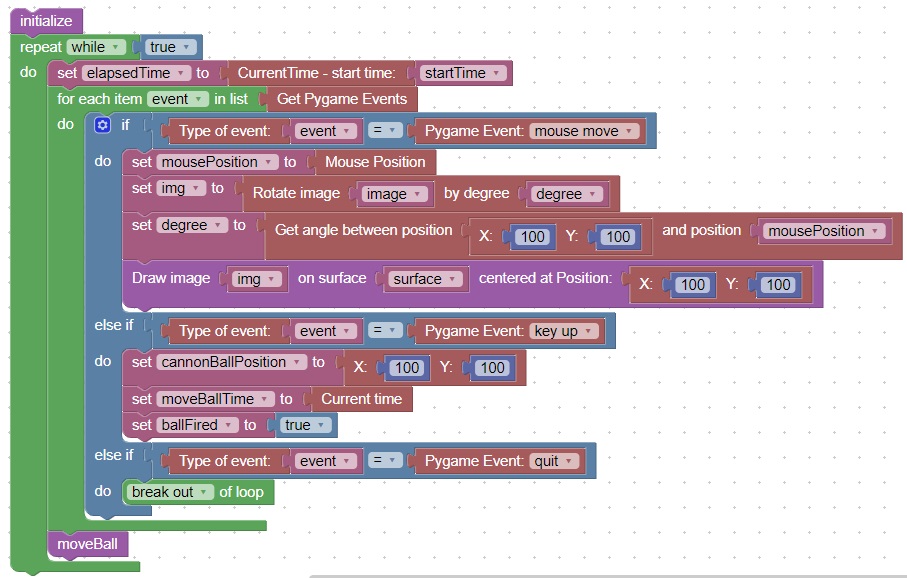
This code will initialize some variables



This code will move the cannonball in the direction of the mouse



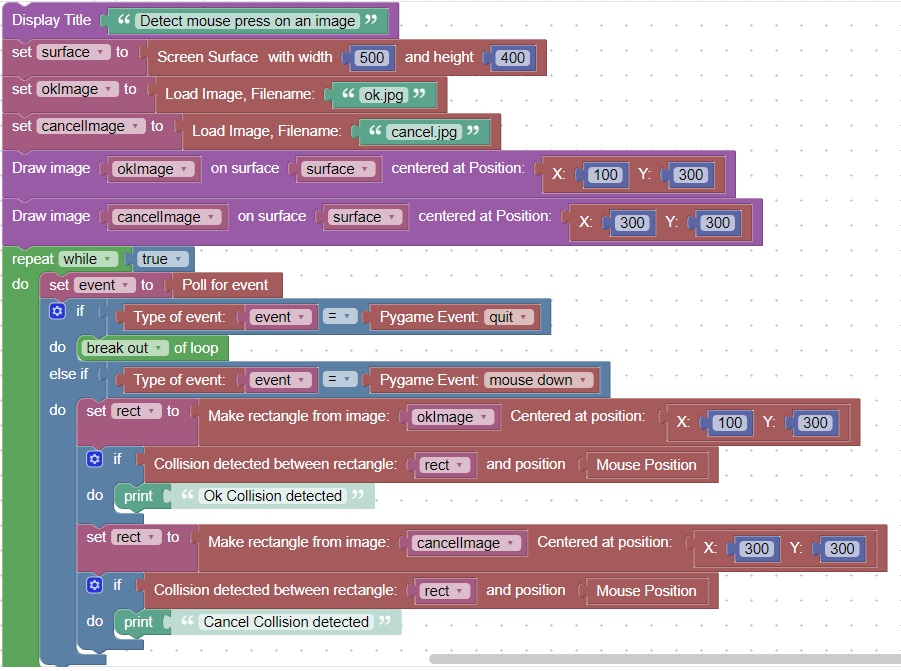
This code will "fire" a cannonball when a button is pressed  
It will stop when the QUIT(windows close) event is triggered



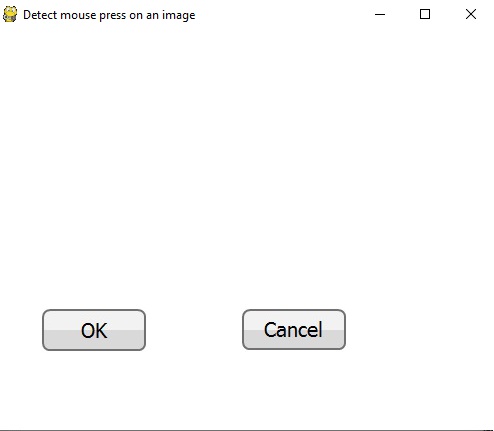
Here is what the screen looks like



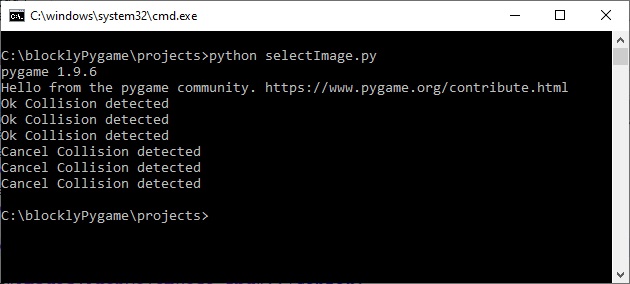
2.4 Collision Detection (Mouse on Image)  
This code will detect a mouse press on an image.



It will look like:

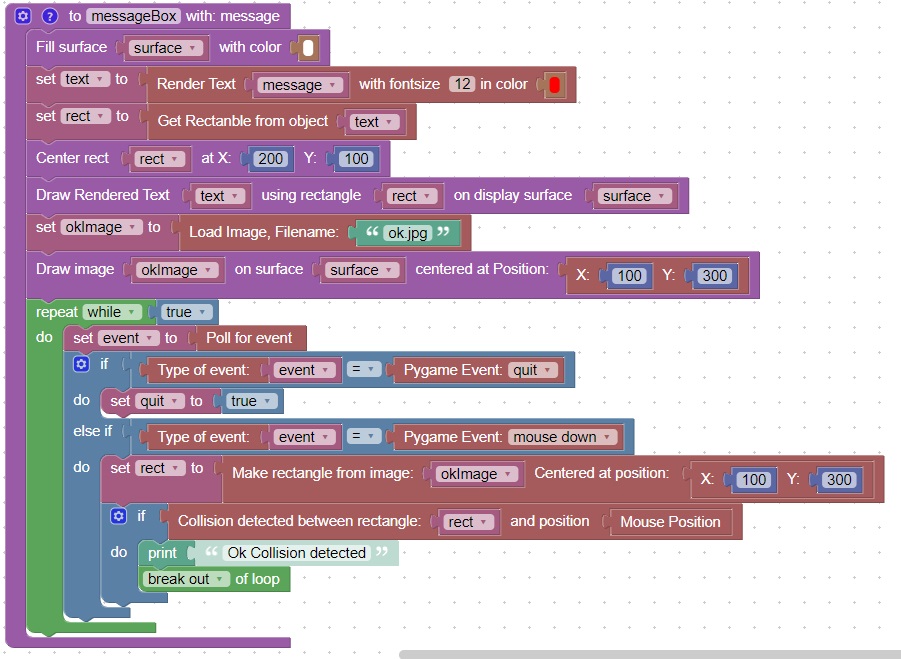


Here is what the console looks like:



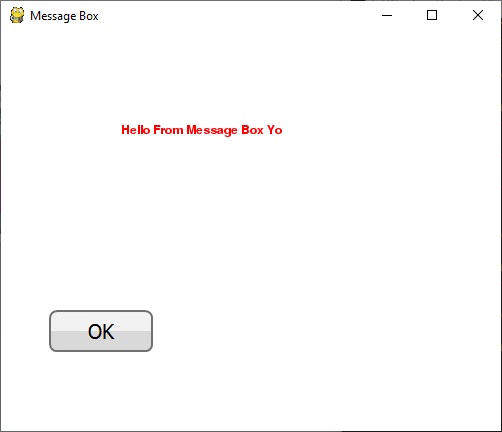
2.5 Message Box

This code will create a messageBox function

  
  
It can be called using this code

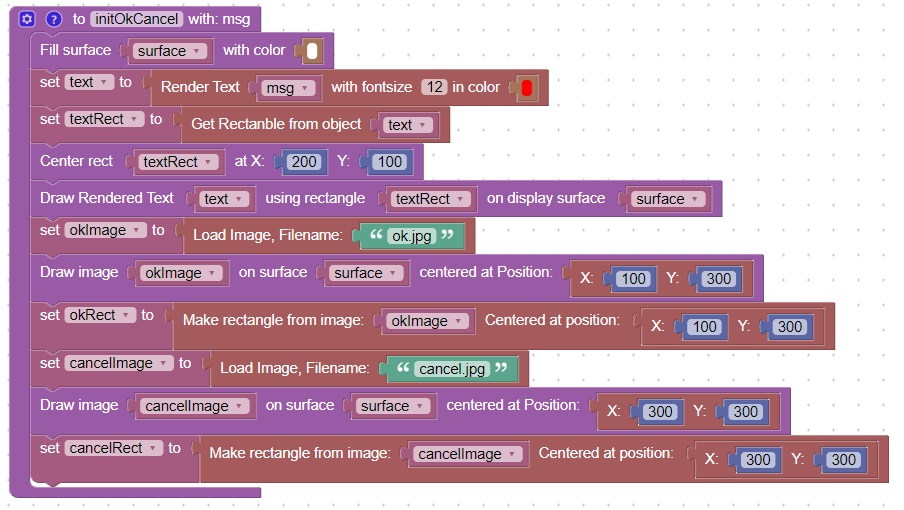


It will appear like this when called:

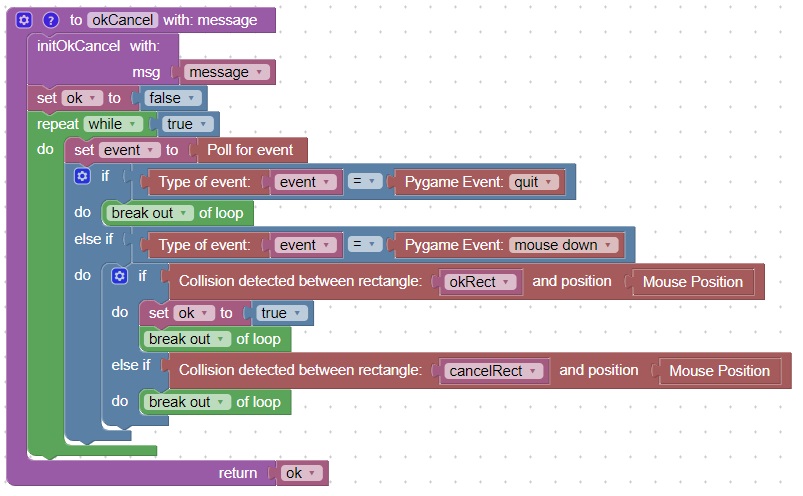


2.6 Ok-Cancel Dialog Box

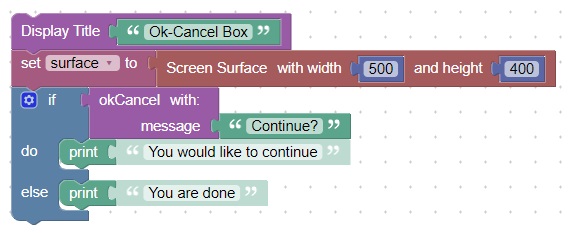
This code will initialize the variables required for the ok/Cancel function

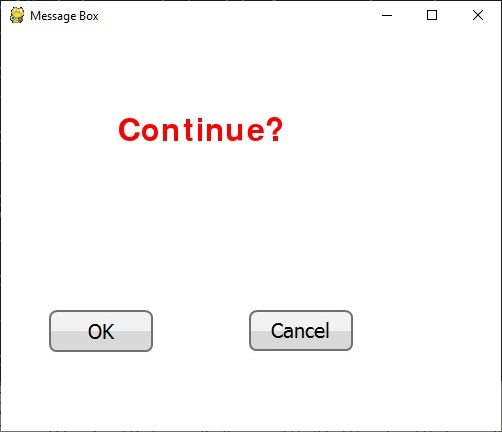


This code will perform an ok/Cancel function

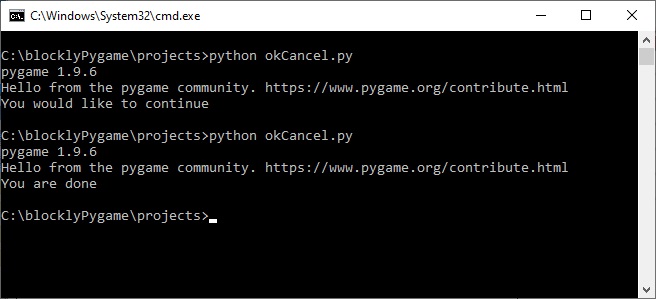


This code will shows an example call of the ok/Cancel function

  
  
This is what the screen looks like when it is executed

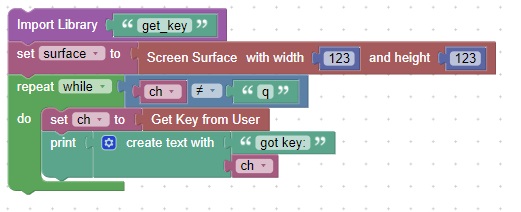


This is what the console looks like when it is run twice and ok is selected the first time, and cancel is selected the second time



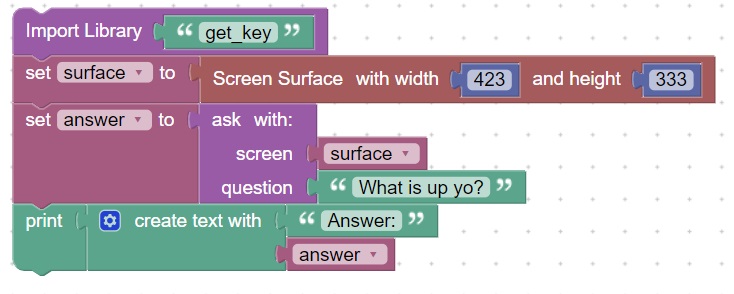
2.7 Read key input from the player

For this code you will need to use a library function called get\_key  
This code will get a key from the user and print it to the console until a 'q' key is pressed.

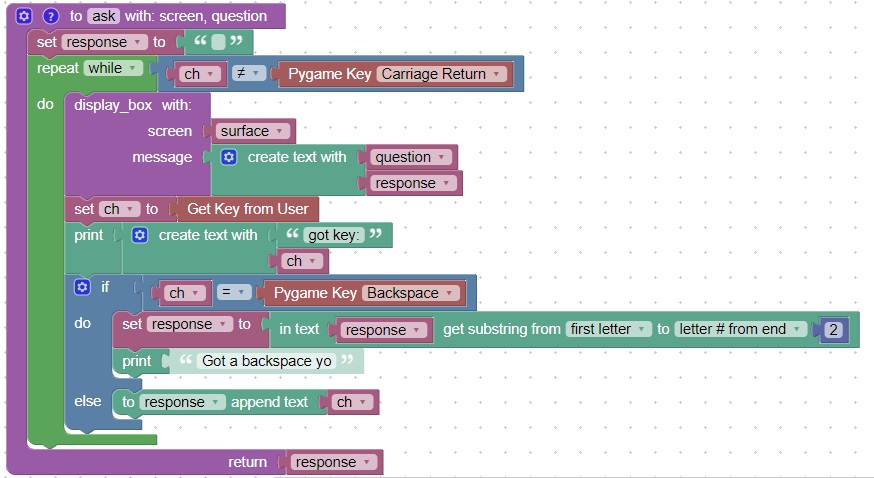


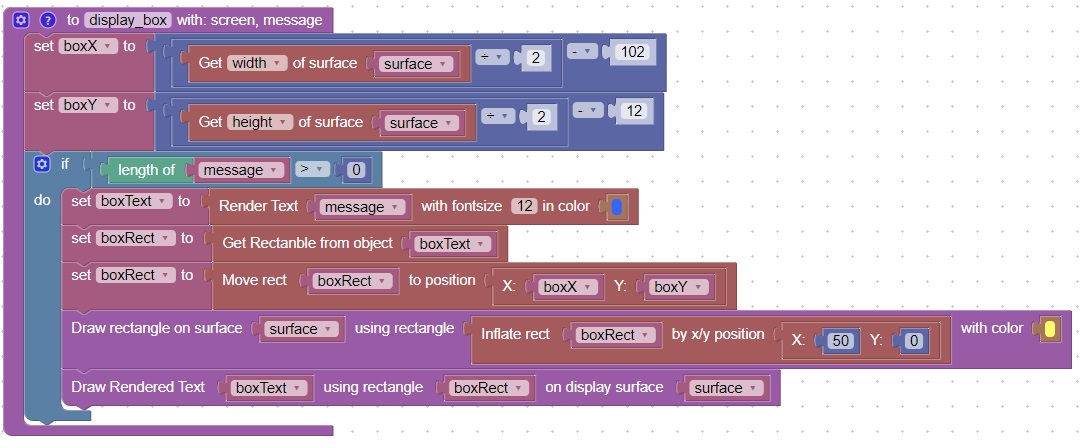
2.8 Input Box

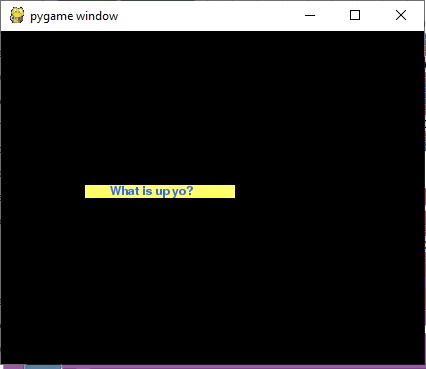
This code will prompt the user to answer a question. It returns the value that was entered.



You will also need 2 functions to complete this task:

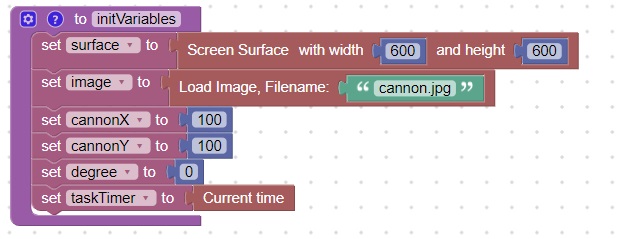
  
  
And

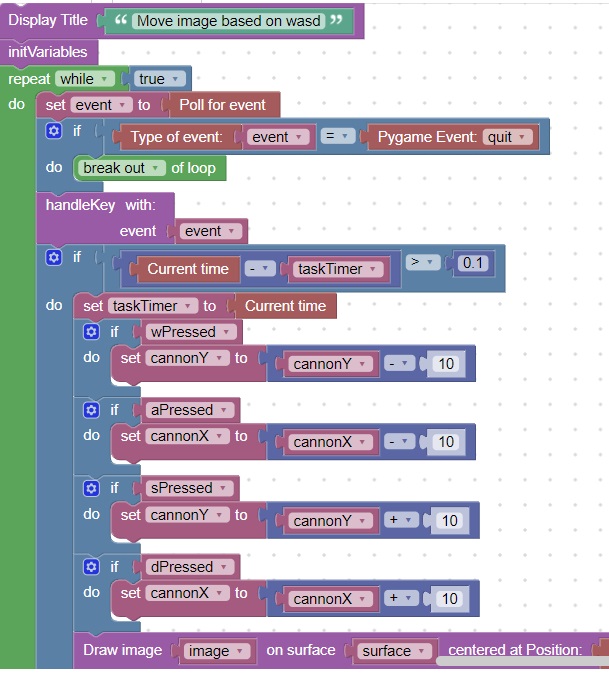
  
  
The screen will look like this:



3.1 Move an image based on wasd keys

This code will initialize the wasdMove example

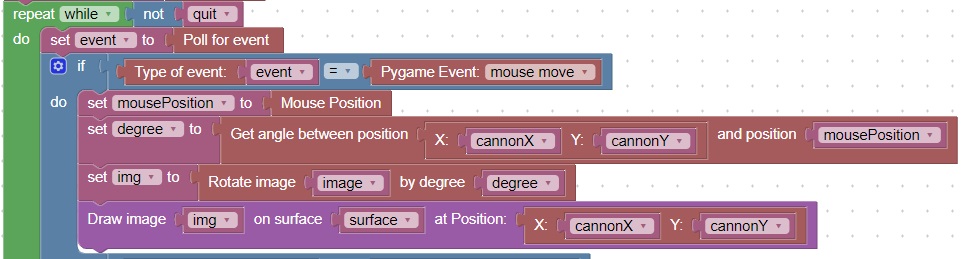
  
  
This code will handle the key press and move the image accordingly

  
  
This code will handle the pygame.KEYUP and pygame.KEYDOWN events and determine the key press



3.2 Move an image based on wasd keys and point towards the mouse

This code will handle the key press and move the image accordingly



3.3 Play a sound on mouse press

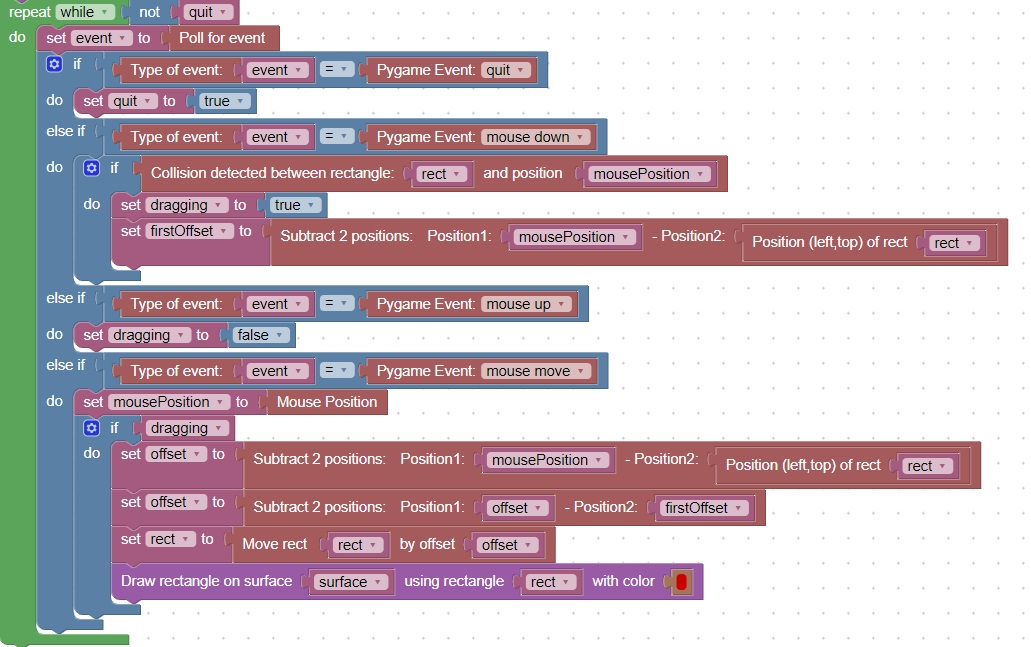
This code will play a sound when the mouse is clicked



3.4 Drag and Drop

This code will initialize the drag and drop rectangle example  

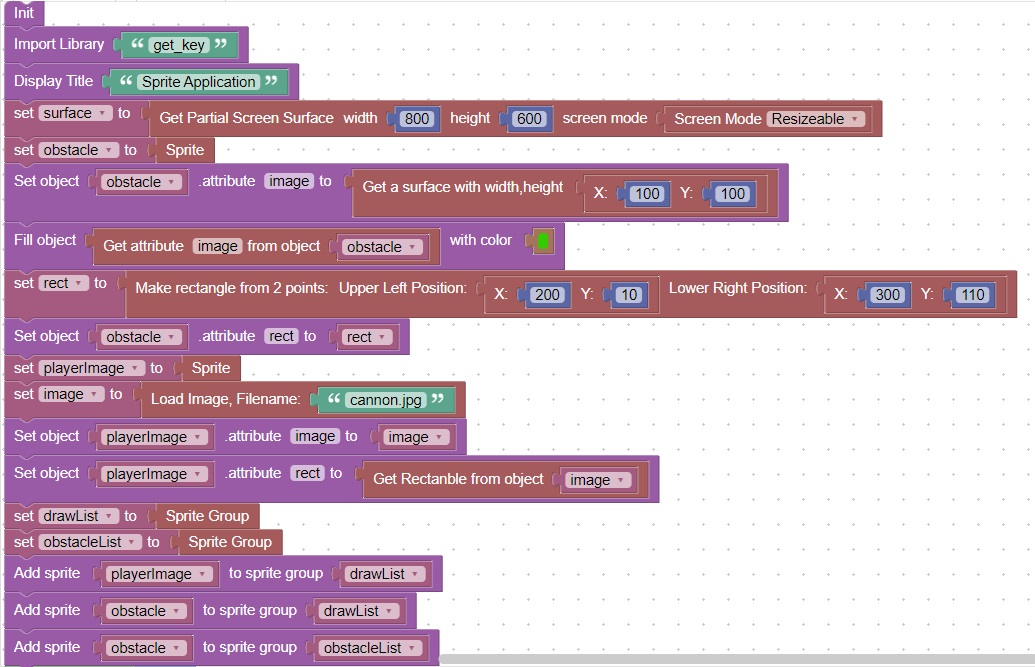

This code will run the main drag/drop loop:



3.5 Sprites  
In this class we will create sprites and draw them on the surface

3.6 Sprite Collisions

This code will initialize the sprite collision example

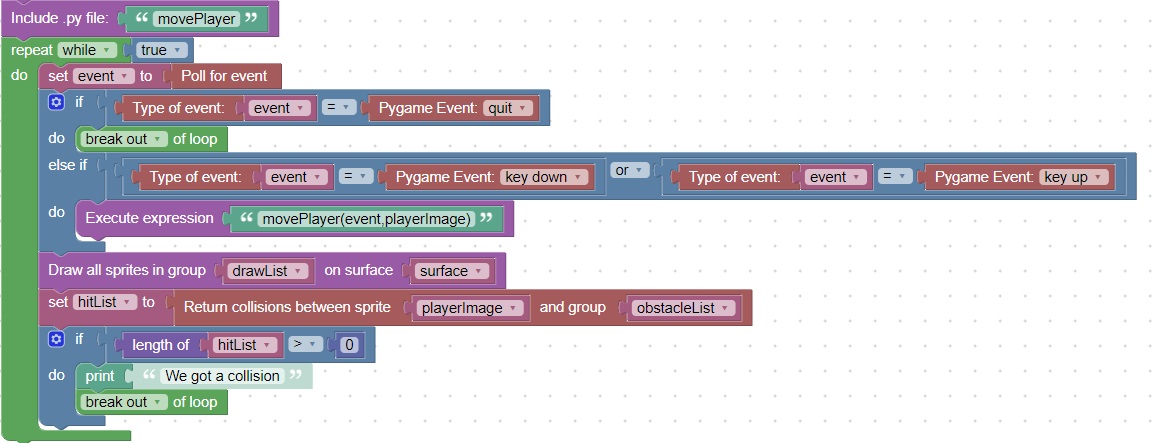


3.7 Call an external .py  
By using the blocks:

Include .py file:

Execute expression

You can include and call an external .py file  
This code will include and call the external .py module: movePlayer.py



3.8 Blasteroids

This game combines all the elements we have learned so far.

Here is the code;

import math

surface = None

obstacle = None

rect = None

playerImage = None

cannonBallImage = None

image = None

img = None

cannonBall = None

drawList = None

obstacleList = None

cannonBallMoveTime = None

event = None

mousePosition = None

position = None

degree = None

hitList = None

import time

import pygame

pygame.init()

from movePlayer import \*

pygame.display.set\_caption('Blasteroids')

surface = pygame.display.set\_mode ((800,600), )

obstacle = pygame.sprite.Sprite()

obstacle.image=(pygame.Surface([((100,100))[0],((100,100))[1]]))

(obstacle.image).fill(pygame.Color('#33cc00'));

rect = pygame.Rect(((200,10)),(((300,110))[0]-((200,10))[0],((300,110))[1]-((200,10))[1]))

obstacle.rect=rect

playerImage = pygame.sprite.Sprite()

cannonBallImage = pygame.sprite.Sprite()

image = pygame.image.load ('ship.jpg').convert()

img = pygame.image.load ('ship.jpg').convert()

cannonBall = pygame.image.load ('cannonBall.jpg').convert()

playerImage.image=img

playerImage.rect=(img.get\_rect())

playerImage.image=cannonBall

cannonBallImage.rect=(cannonBall.get\_rect())

cannonBallImage.image=cannonBall

cannonBallImage.deltaX=0

cannonBallImage.deltaY=0

drawList = pygame.sprite.Group()

obstacleList = pygame.sprite.Group()

drawList.add(playerImage)

drawList.add(obstacle)

obstacleList.add(obstacle)

cannonBallMoveTime = time.time()

while True:

event = pygame.event.poll()

if (event.type) == (pygame.QUIT):

break

elif (event.type) == (pygame.KEYDOWN) or (event.type) == (pygame.KEYUP):

if (event.type) == (pygame.KEYDOWN) and (chr(event.key)) == ' ':

cannonBallImage.deltaX=(math.cos(degree / 180.0 \* math.pi) \* 10)

cannonBallImage.deltaY=(math.sin((degree + 180) / 180.0 \* math.pi) \* 10)

cannonBallImage.rect.y=(playerImage.rect.center[1])

cannonBallImage.rect.x=(playerImage.rect.center[0])

drawList.add(cannonBallImage)

print("deltaX: " + str(cannonBallImage.deltaX))

print("deltaY: " + str(cannonBallImage.deltaY))

else:

movePlayer(event,playerImage)

surface.fill (0Xffffff)

pygame.display.update()

drawList.draw(surface)

pygame.display.update()

hitList = pygame.sprite.spritecollide(playerImage,obstacleList,False)

if len(hitList) > 0:

print('We got a collision')

break

elif (event.type) == (pygame.MOUSEMOTION):

mousePosition = pygame.mouse.get\_pos()

position = (playerImage.rect).center

degree = math.degrees(math.atan2(-(mousePosition[1] - position[1]),mousePosition[0]-position[0]) % (2\*math.pi))

playerImage.image=(pygame.transform.rotate(image,degree))

drawList.draw(surface)

pygame.display.update()

elif (time.time() - cannonBallMoveTime) > 0.01:

cannonBallMoveTime = time.time()

cannonBallImage.rect.x=((cannonBallImage.rect.x) + (cannonBallImage.deltaX))

cannonBallImage.rect.y=((cannonBallImage.rect.y) + (cannonBallImage.deltaY))

drawList.draw(surface)

pygame.display.update()