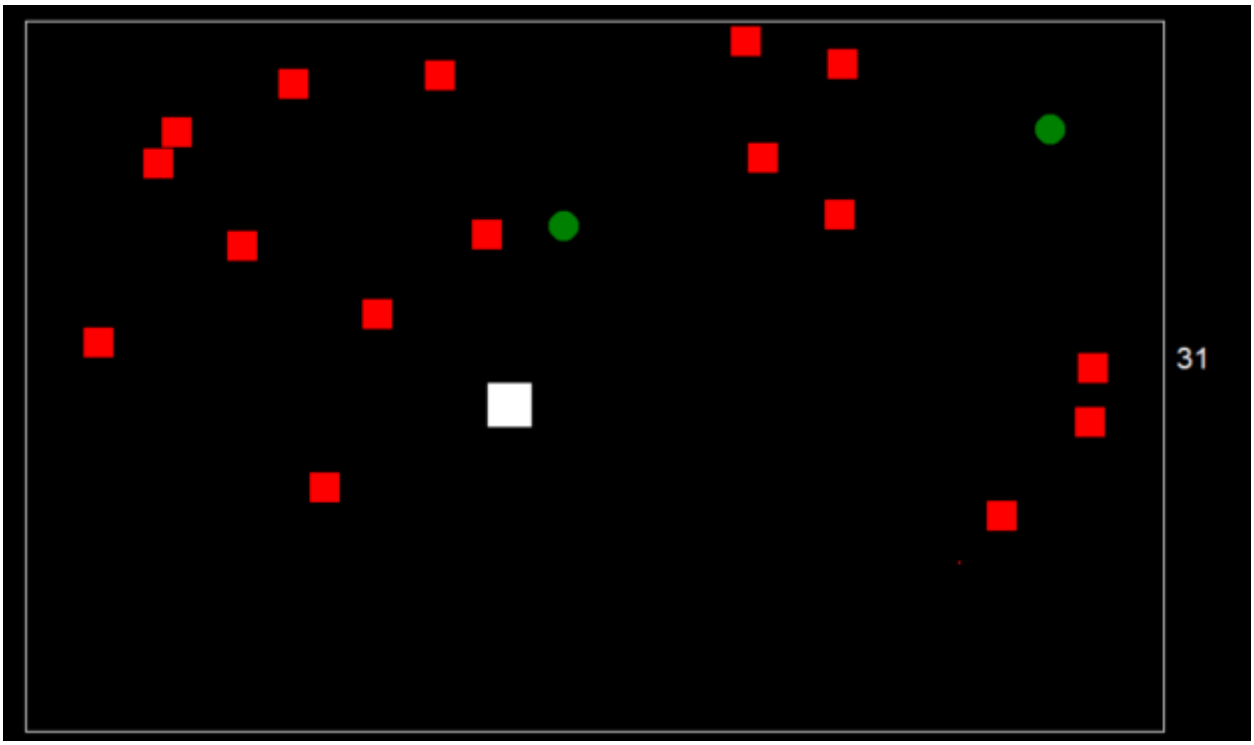


Block Avoider Project Instructions!

WALK THROUGH VIDEO

Step 0. Create a new file [0:39 in video]

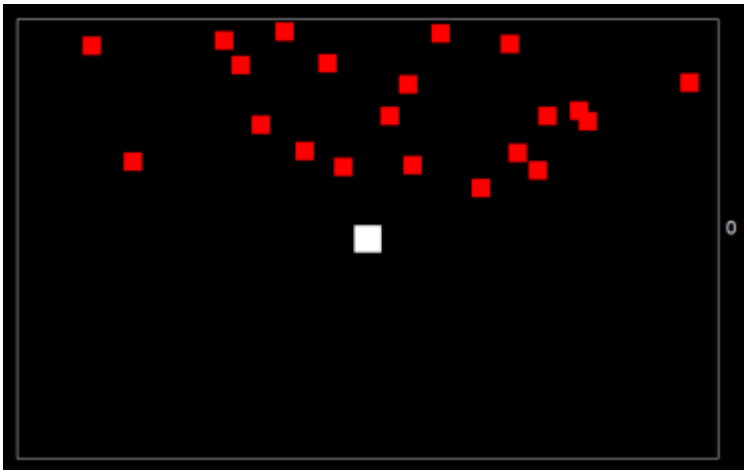
- On the computer, open up Virtual Studio Code
- Create a new file.
- Hit ctrl + s (Save the file). Make sure to save it in the desktop folder “Python Code”.
- INCLUDE YOUR NAMES IN THE FILE NAME!!!
- Go to the project folder and copy and paste the starter code into your file. Save!
- Run the file to make sure that there are no errors!

Step 1. Draw a border [2:07 in video]

- Complete the make_border function

```
#STEP 1 - Make the game border
def make_border():
    '''
    makes a white border around the game screen
    using variables WIDTH and HEIGHT
    '''
    turtle.color("white")
    turtle.penup()
    turtle.hideturtle() #hides the arrow
```

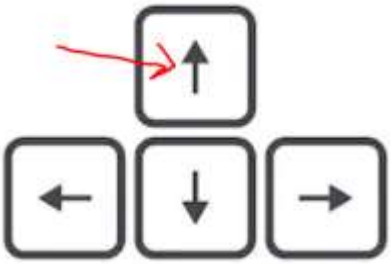
- When you are done, hit run. It should draw a box like this (but bigger):

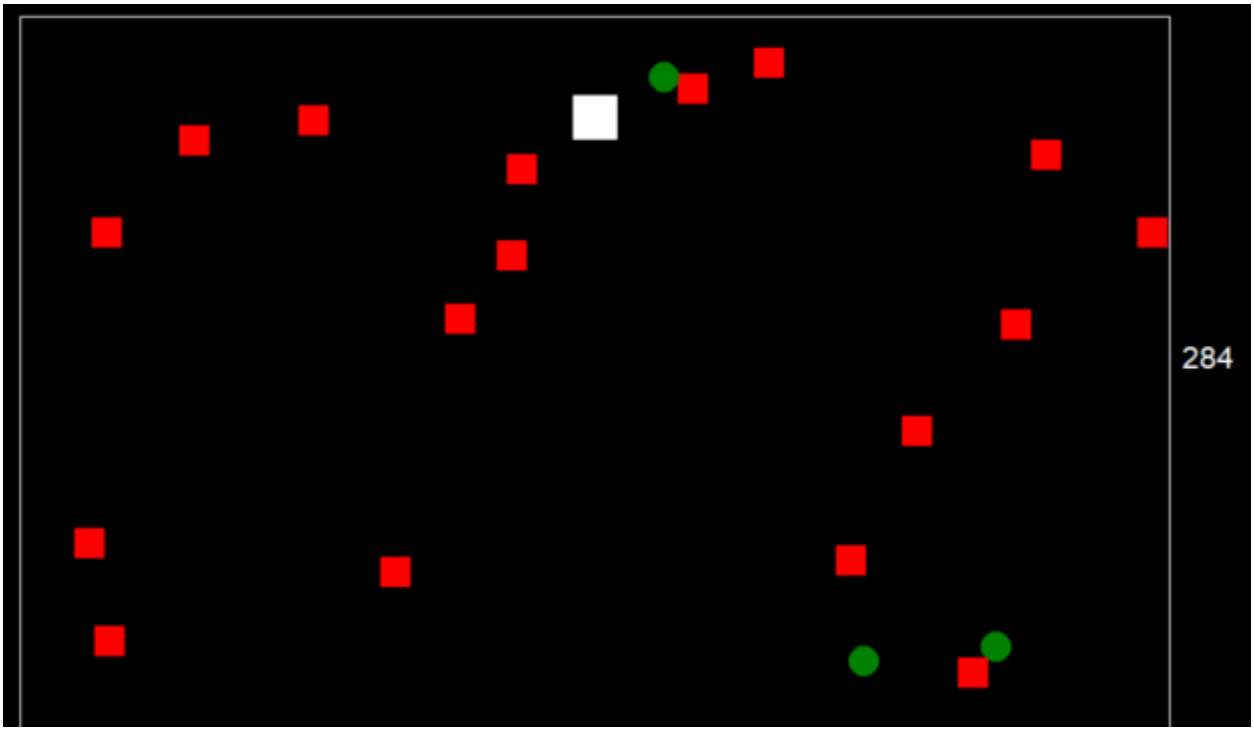


- Check for understanding: what happens if you change WIDTH = 100? (instead of 250-- it should be around line 21)

Step 2. Move around! [2:51 in video]

- Run the program. If you hit the up arrow, the player (the white box) should move up:





- Complete the left(), right() and down() functions so that the pen can move in all four directions!
    - HINT : The following powerpoint is useful
    - HINT: Use the up() function as a model:

```
#STEP 2 - Move the player around
def up():
    '''
    Turns player North and moves it up by distance amount
    '''
    player.setheading(90) # faces up
    player.forward(distance) #move turle forward by distance

turtle.onkeypress(up, "Up") # Create listener for Up key
```

STEP 3. Keep the player inside the walls [4:07 in video]

- Complete the can_move() function

```
#STEP 3 - Player stays in border - complete the can_move() function
def can_move(x,y):
    '''
    x - int, x coordinate of player
    y - int, y coordinate of player
    return True if position is inside the border,
    False if it is outside the walls
    '''
    result = True
    #your code here
    if x > WIDTH:
        result = False
    if x < -WIDTH: #left wall
        result = False
    if y > HEIGHT: #top wall
        result = False
    if y < - HEIGHT: #bottom wall
        result = False
    #end your code
    return result
```

- Then update your up(), down(), left(), and right() function to check IF the player can_move() before moving.
- Hint: the can_move function is VERY similar to W2D2 Pet Turtle assignment...
- Hint: This is what the up function should look like afterwards

```
def up():
    '''
    Turns player North and moves it up by distance amount
    '''
    x = player.xcor() # new x coorindate of player
    y = player.ycor() + distance # new y coordinate of player
    if can_move(x,y):
        player.setheading(90) # faces up
        player.forward(distance) #move turle forward by distance

turtle.onkeypress(up, "Up") # Create listener for Up key
```

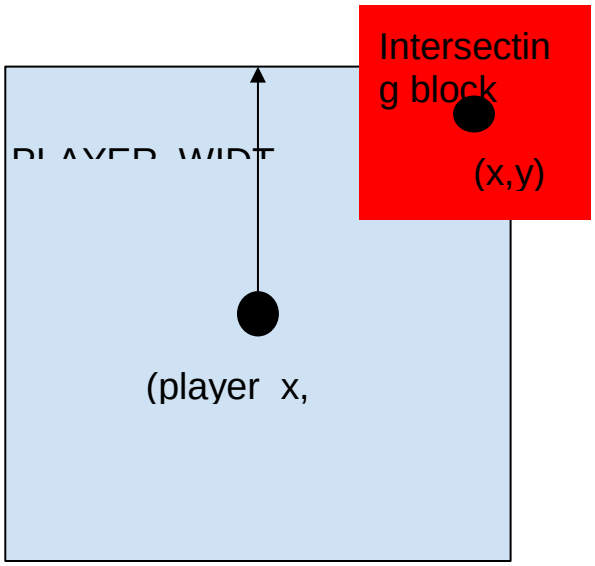
STEP 4. Check if player is touching blocks [6:30]

- Complete the isTouching function

```
#STEP 4 - Check if player is touching falling objects
def isTouching(x,y):
    '''
    x - int, x coordinate of falling object
    y - int, y coordinate of falling object
    return True if position is inside the player, False otherwise
    '''
    player_x = player.xcor()
    player_y = player.ycor()

    result = False
    #your code here

    #end code
    return result
```

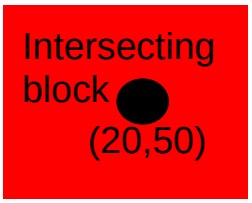


The player has its center at (player_x, player_y).

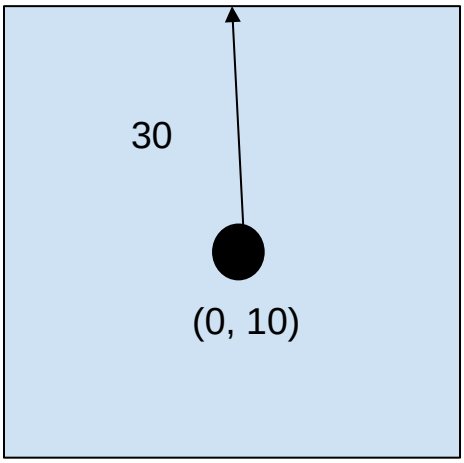
The block isTouching player When both the x and the y Of the block are within PLAYER_WIDTH distance of player_x, player_y

For example:
player_x = 0
player_y = 10
PLAYER_WIDTH = 30
If I have a block with:
X = 20
And Y = 50
Is it touching player?

No!



We can tell by taking
The absolute value
Of the difference between
the two.



Difference in x = |x - player_x|
Difference in x = | 20 - 0 |
Difference in x = 20

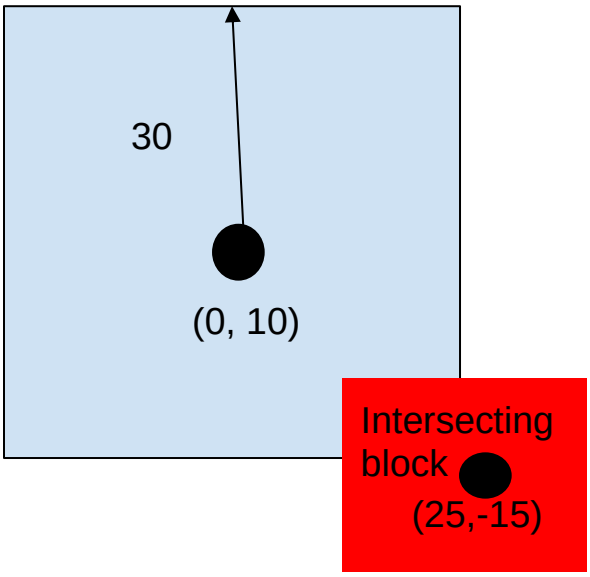
So that is within PLAYER_WIDTH (30) of the center.

But what about y?

Difference in y = |y - player_y|
Difference in y = | 50 - 10 |
Difference in y = 40
That is greater than PLAYER_WIDTH (40 > 30) so the blocks are not intersecting.
Let's take another example:
player_x = 0
player_y = 10
PLAYER_WIDTH = 30
If I have a block with:
X = 25
And Y = -15

Difference in x = |x - player_x|
Difference in x = | 25 - 0 |
Difference in x = 25

Difference in y = |y - player_y|
Difference in y = | -15 - 10 |
Difference in y = 25



Both differences are 25, which is less than the PLAYER_WIDTH of 30 so the two blocks are intersecting!

But how do we do this in code?

We can take the absolute difference of two numbers by using the abs() (short for absolute) function
Ex.

abs(-15) → results in 15

abs(-15-10) → results in 25

abs(10) → results in 10

- Using the abs() function, find the difference in x and difference in y. IF the difference in x AND y are less than PLAYER_WIDTH, result should be set to True.

If you need help coding -- watch the video!

- When you are done blocks should disappear if you intercept them!

STEP 5. Subtract lives and game over [9:07]

- Complete the check_player function to subtract lives if the player is touching a spike (red block)

```
#checks if player has been hit by falling spike
def check_player():
    '''
    For all blocks / spikes, check if player is touching it.
    If so, subtract a life from num_lives.
    If player reaches 0 lives, game is over.
    '''
    global num_lives
    for t in spikes:
        x,y = t.pos()
        if isTouching(x,y): #player is hit by blocks
            t.hideturtle()
            t.goto(x, -MAX_Y)
            #your code here
```

- Modify the game_over() function

```
def game_over():
    '''
    When the game ends
    - change player color to red
    - pause for 5 seconds
    - and quit the screen
    '''
    #your code here
    player.color("red")
    time.sleep(5)
    quit()  #Exit screen
```

- If num_lives is 0, call the game_over() function in game()!

```
def game():
    '''
    This function controls the game!
    It moves the falling objects down, checks if the player has
    been hit, and adds difficulty as the game progresses.
    '''
    for t in spikes + fruits:
        x = t.xcor()
        y = t.ycor()
        t.goto(x,y-DISTANCE) # move spikes and fruits down
    check_player() # check if player is hit by falling blocks
    check_fruit() #check if player ate fruit
    check_edge() # check if falling objects hit the bottom wall
    update_score() #write new score
    # STEP 5 YOUR CODE HERE


    #end your code
    turtle.ontimer(game, TIME_STEP) #after TIME_STEP amount of time, repeat
```

See the video for hints!

STEP 6. Add your own features! [11:49 in video]

- *Make sure to save a separate copy of your working code first in case something breaks!!!*

Ideas:

- Add a livesWriter that writes how many lives the player has remaining. (hint look at the scoreWriter code at the bottom of the program. Note how the game() function calls updateScore!)
- Can you figure out how to increase the difficulty as the score increases?
- Can you let the user select how fast the game speed is at the beginning of the game? (use input function and change TIME_STEP)
- How could you make the game pause when you hit the space bar?
- Can you make a GAME OVER screen?

STEP 7. MAKE SURE YOU FINISH THE DESIGN REFLECTION AND SUBMIT YOUR CODE!!!!!!!

If you finish:

- Have you submitted all of your assignments?
- Have you done test corrections?
- If you answered yes to both of the above, feel free to either help other groups (do not write code for them), add more features to your project or do the other project for extra practice (and extra fun!)