17085721 Caverns-Ai For Games Development

Clickable Table Of Contents

Scenario Analysis:

The game we have chosen is Caverns, in the game the player collects different types of fruit and health upgrades (+1),that give him points for his score in order for the player to proceed to the next level he needs to kill off all the enemies in the level, the fruit and health upgrades have a limited amount of life existence in the level so the player needs to collect as many as possible before they disappear they spawn in random positions and at random times there's a limited amount of them in each level.

Normally the player can move left , right , jump up on the platforms he can fire bubbles at the enemies by pressing space and the player has the option to drop through openings at the bottom of the screen that will spawn him of the top of the screen on the top platforms.

The game will end when the player has no more lives, and he is total high score will depend on how many fruit and the types of fruit the player collected and how many levels the player managed to progress through.

The enemies can fire bullets in order to reduce the player lives , the player can defend against the bullets and enemies by firing bubbles at them but have a minimum range before the bubbles start flowing up alternatively the player can choose to evade the enemies that he meets on the platforms by jumping to a lower platform or an upper platform.

Now the enemies move only on the direction they have spawned in, for example if they spawn looking left they will keep heading left until either they hit a wall which then they will be forced to head right or if the player is close enough to them they will change direction in order to fire upon the player.

The enemies fire at random without the player needing to be in the vicinity and they fire more often at higher levels also at higher levels there are more enemies.

So, lets break this down into a simpler format:

- The player can move left, right, or jump onto the platforms moving down by dropping from one platform to another.
- The player can only progress through a level if he kills all the enemies.
- The player collects fruit that spawn randomly on the level and there is only a certain amount of them to collect points (Different Fruit are worth different amount of points).
- For the player to fire upon an enemy and kill him he needs to stop and face the enemy before the player fires a bubble.
- Fruit have a time limit of existence.
- The enemies can reduce players life's by firing at the player.
- The enemies fire at random and the more levels the player progresses through the more enemies there are and more frequently they fire.
- The player can collect lives to stay in the game but those are spawned randomly at random times and levels.
- The game end's when the player runs out of lives.
- The end goal is to reach the highest level possible with the highest amount of points possible.

Ai Choice:

FSM:

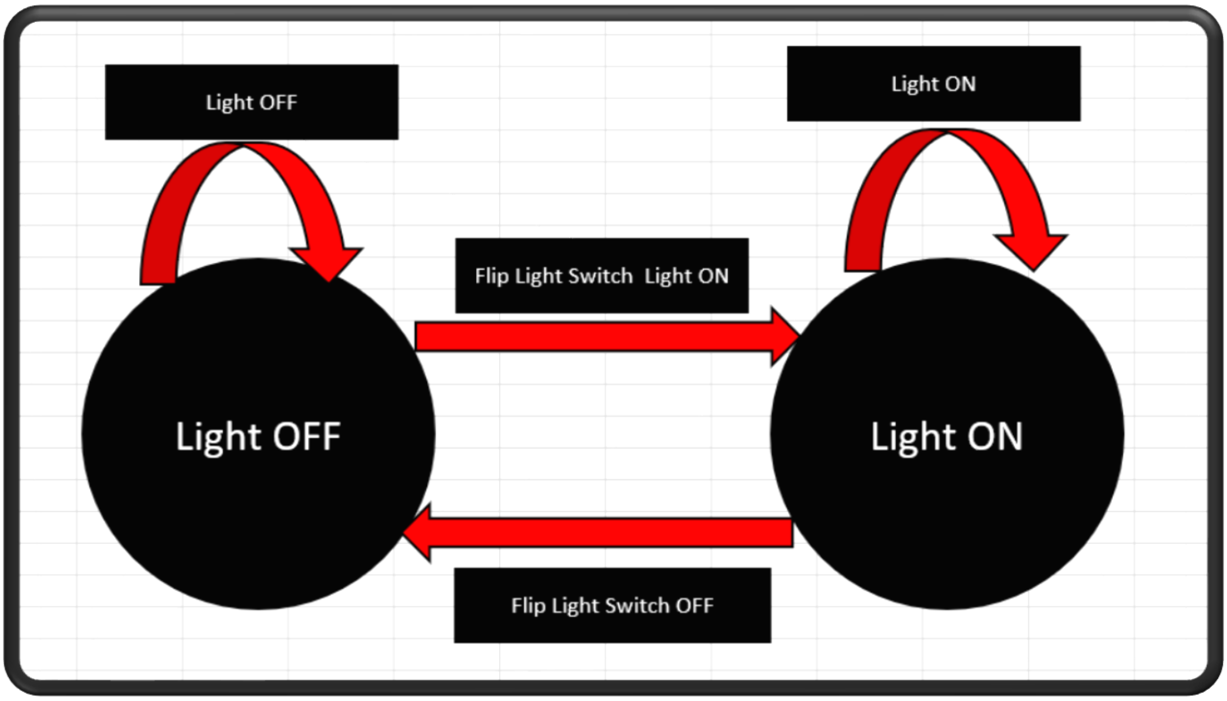
A finite stat machine is an abstract machine that consists of different states and input events that are handled by transition functions.

It takes the current state and an input event and returns the new set of output events and the next (new) states.

It is great for implementing Ai in computer games.

Since though only one state can be active at a time it needs to transition through these states to perform different actions.

So let's thing of a simple FSM as an example a light bulb in a room can only have 2 states at a time either On or Off the transitions to this states happens when you flip the switch you flip the switch if the light was off then it turns on you flip It again it turns off here is a simple diagram.



Now you might notice that we loop on this states that's because if you think again of our light example if you flip the switch On and you keep pressing down on the switch while it's in the ON state the light will stay on emitting light only when you flip it to the off positions we will transition to the off state.

The drawback of applying an FSM is that you can have a limited amount of states, you can off course have as many as you like but at some point it will be overwhelmed by the total amount of states you have implanted.

Why FSM And Not A* For Caverns:

The reason why FSM is better to be implemented for Caverns is because our player doesn't only need to move from one point to another avoiding obstacles until he reaches the end node our player has different states that he can be in at any given time and it doesn't really matter to follow an optimal route rather it needs to truck where the positions of the fruit, the enemies or if it's under attack or not (Incoming projectile) like a normal player will do if he is playing the game what you see is what you can do the only difference will be that our ai for the sake of not been perfect so it can lose will only truck the object that is first in the lists been fruit, enemy or projectile sort of like a FIFO first in first out it will only be able to work with the first things in those lists and change states accordingly.

Also, the amount of states is minimal so we will not have a big FSM rather a simple one that would get the job done.

Also, A* will need to check our whole graph to find the most optimal routes to sed object's which we do not actually need necessarily for this implementation.

Scenario Solution:

We have chosen an FSM this is our not implemented solution rather the one we used in cavern-Ai 1.0:

1) Collecting Fruit/Life
2) Attacking
3) Defending
4) Death

Let us have a closer look at this state's:

Collecting Fruit/Life state:

In this state our Ai will go about collecting the fruit in the level we are applying a FIFO technique us it will go for the first fruit that spawns in the game ones that is removed the second one will take its place so first fruit that spawns in our list position [0] we go after it.

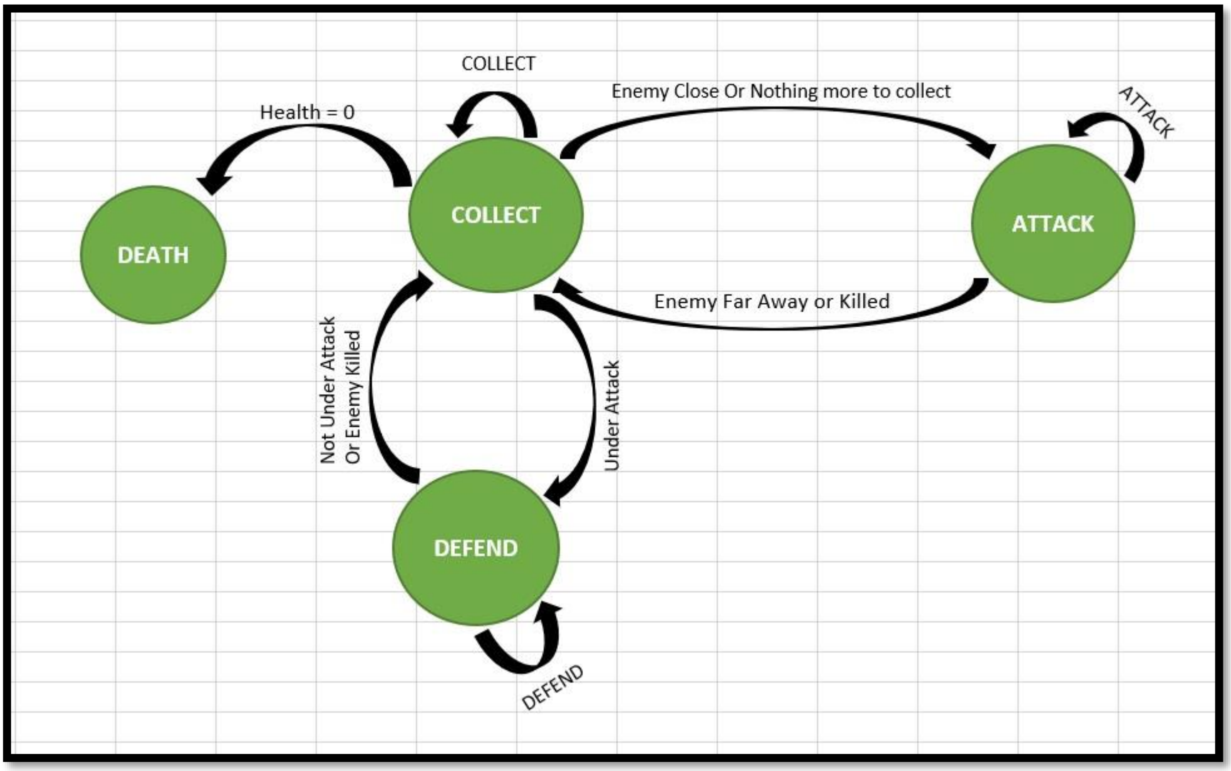
Attacking State:

Our Ai player can be in an attacking state if the enemy is close to him when collecting fruit it can switch from the collecting fruit state and attack the enemy (fire its bubbles) at the closest enemy or lock in to an attacking state if there are no more fruit to collect thus needing to eliminate the enemy's to progress to the next level.

Defending State:

Our Ai can enter a defending state if at any point its going around collecting its points it comes under attack by enemy fire or an enemy if it happens our Ai can use its bubbles to deflect the enemy bullets or kill the enemy that it's trying to collide with it.

Death State: Our Ai will enter the death state effectively ending the game when it runs out of lives meaning the enemies managed to kill the player off.



Here is our FSM diagram for the above states lets go through it together shall we.

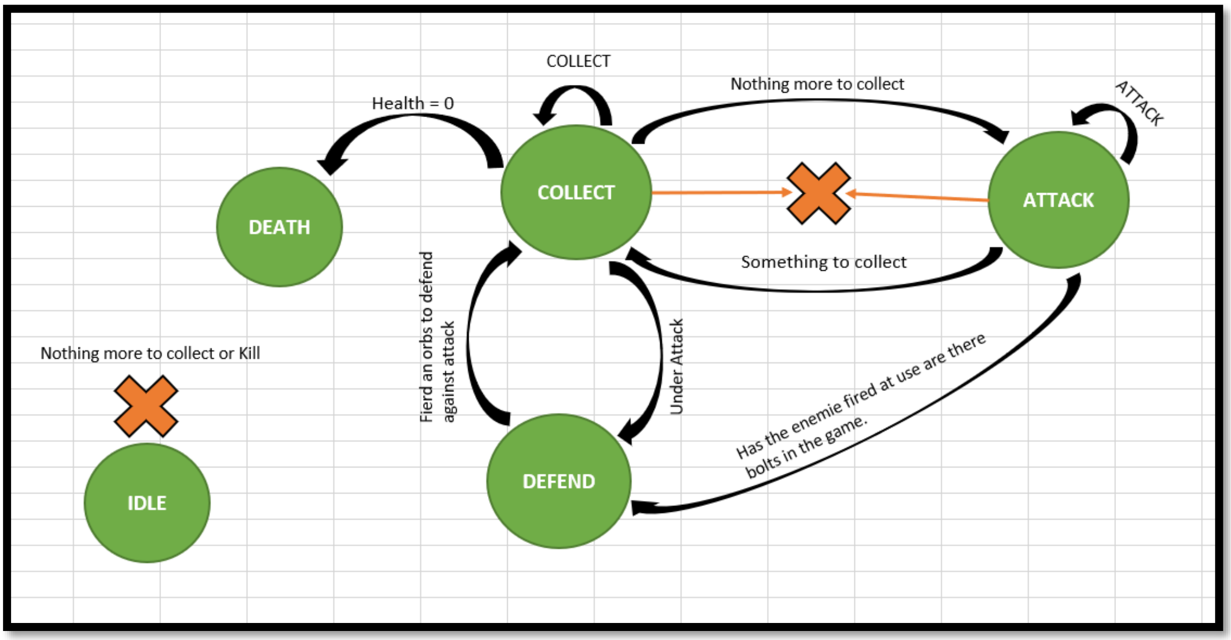
Our Ai's beginning state is the Collect state where it will go around collecting fruit and +1 health it will stay in the collect state until there is nothing more to collect or its under attack or an enemy is close to the Ai.

If an enemy is close to the Ai or there's nothing more to collect we will move into an attack state and we will remain there until the enemy is killed or managed to move away from the player if that is the case we will move back to our collect state.

In the case our Ai comes under attack we will move to the defend state and stay there until either we are not under attack anymore meaning it defended against the bullets and the enemies moved on or it killed the enemy that was attacking it.

The only way for AI to enter the death state is if its health has dropped to zero meaning it was killed effectively ending the game.

Implemented Solution:


COLLECT
Nothing more to collect
ATTACK
Health = 0
DEATH
COLLECT
ATTACK
Something to collect
Fierd an orbs to defend against attack
Under Attack
Nothing more to collect or Kill
Has the enemie fired at use are there bolts in the game.
IDLE
DEFEND

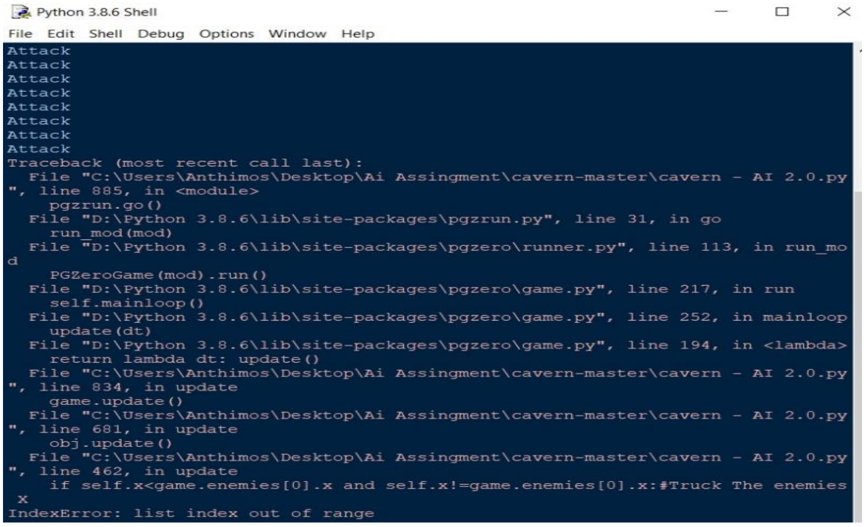
This is what we managed to implement not so different than our first solution but close enough what changed was our conditions and we introduced a new state.

Rather than checking if an enemy is close in order to attack we check if there are no fruit to collect and enter the attack state we will stay there until there are fruits to collect if not will just hunt down the enemies in the case that the enemy fires at us and we are on the same Y position as the bolt we will transition too our defend state.

Our death state will function the same if our health drops to 0 we die and end the game.

If a bolt is fired at us and we are on the same Y location with that bolt, we will defend our self's by firing orbs to the direction of the bolt then we will revert back to our collect state regardless if we defended successfully or not.

We have also introduced a new idle state that the player will just stay there In the case that all enemies and fruit has been collected this is to avoid list index out of range errors like the one bellow.

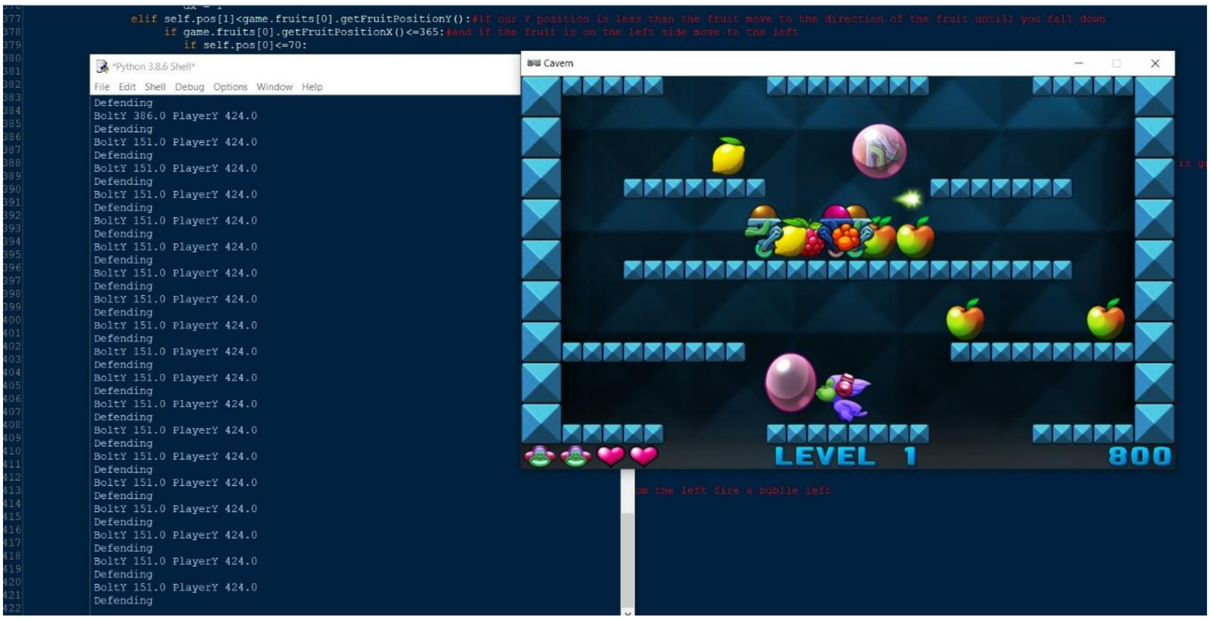


```
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Traceback (most recent call last):
  File "C:\Users\Anthimos\Desktop\Ai Assingment\cavern-master\cavern - AI 2.0.py
", line 885, in <module>
    pgzrun.go()
  File "D:\Python 3.8.6\lib\site-packages\pgzrun.py", line 31, in go
    run_mod(mod)
  File "D:\Python 3.8.6\lib\site-packages\pgzero\runner.py", line 113, in run_mo
d
    PGZeroGame(mod).run()
  File "D:\Python 3.8.6\lib\site-packages\pgzero\game.py", line 217, in run
    self.mainloop()
  File "D:\Python 3.8.6\lib\site-packages\pgzero\game.py", line 252, in mainloop
    update(dt)
  File "D:\Python 3.8.6\lib\site-packages\pgzero\game.py", line 194, in <lambda>
    return lambda dt: update()
  File "C:\Users\Anthimos\Desktop\Ai Assingment\cavern-master\cavern - AI 2.0.py
", line 834, in update
    game.update()
  File "C:\Users\Anthimos\Desktop\Ai Assingment\cavern-master\cavern - AI 2.0.py
", line 681, in update
    obj.update()
  File "C:\Users\Anthimos\Desktop\Ai Assingment\cavern-master\cavern - AI 2.0.py
", line 462, in update
    if self.x<game.enemies[0].x and self.x!=game.enemies[0].x:#Truck The enemies
 x
IndexError: list index out of range
```

Testing Of States:

DEFFEND:

We have tested our different states in order to check that we are entering them bellow is a test run for the defend state at this point we where printing out where the bolt where and where the player was, whenever a bolt was fired we will enter the state in order to make sure that we transition to it.

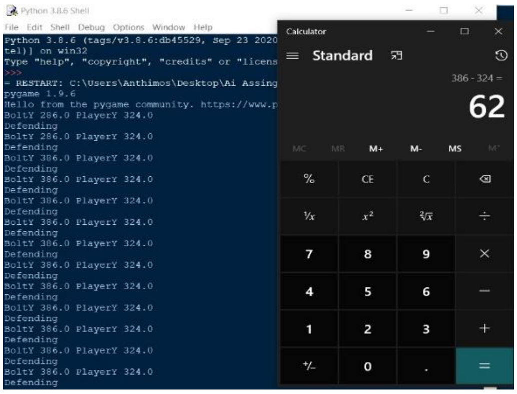


```
            elif self.pos[1]<game.fruits[0].getFruitPositionY():#If our Y position is less than the fruit move to the direction of the fruit untill you fall down
                if game.fruits[0].getFruitPositionX()<=365:#and if the fruit is on the left side move to the left
                  if self.pos[0]<=70:
```

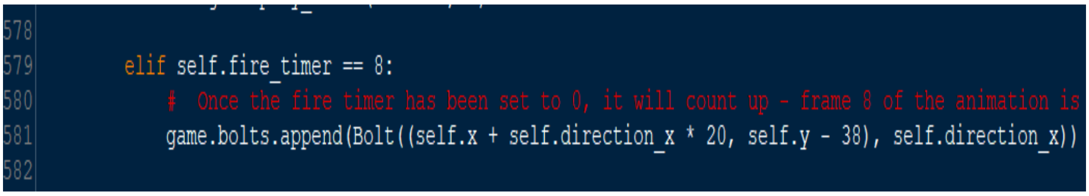
```
Defending
BoltY 386.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
BoltY 151.0 PlayerY 424.0
Defending
```

To make sure we will only enter this state if a bolt was on the same Y with us, I calculated the average between the player and the bolt.

This was no good as actually the solution was a lot simpler.



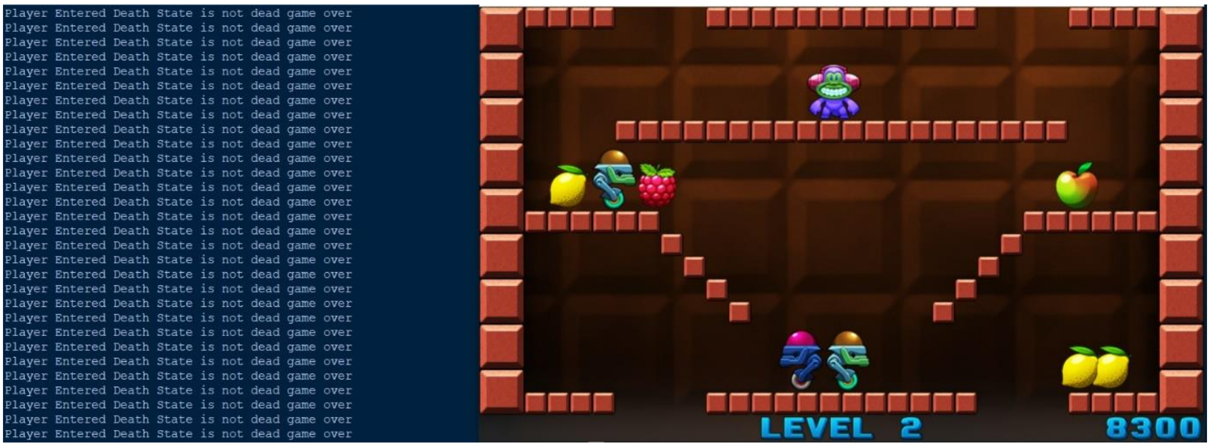
```
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020
tel)] on win32
Type "help", "copyright", "credits" or "licens
>>>
= RESTART: C:\Users\Anthimos\Desktop\Ai Assing
pygame 1.9.6
Hello from the pygame community. https://www.p
BoltY 286.0 PlayerY 324.0
Defending
BoltY 286.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
BoltY 386.0 PlayerY 324.0
Defending
```



```
        elif self.fire_timer == 8:
            #  Once the fire timer has been set to 0, it will count up - frame 8 of the animation is
            game.bolts.append(Bolt((self.x + self.direction_x * 20, self.y - 38), self.direction_x))
```

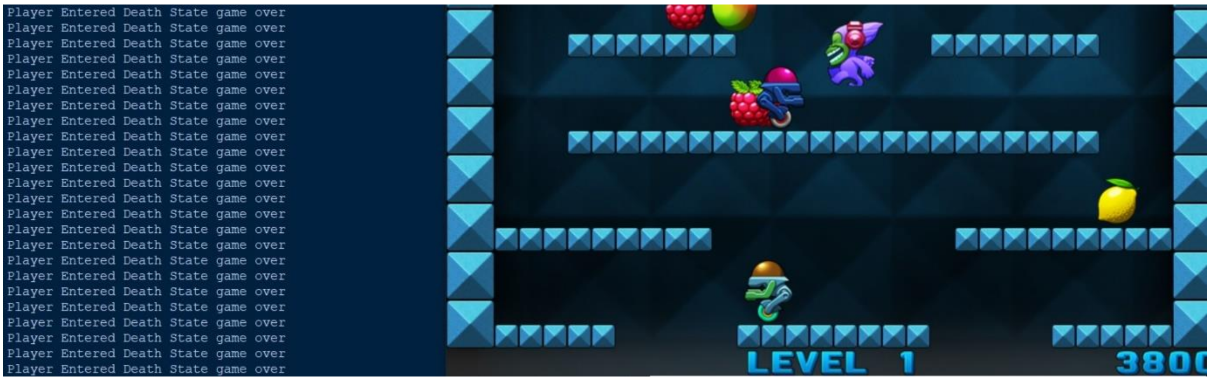
After a quick look at the class robot we could see that the bolt was spawned -38 of the robots Y position so what we actually needed to do was just subtract -38 from the players why position if it was the same as the bolt then it meant the bolt was heading straight to us.

DEATH:



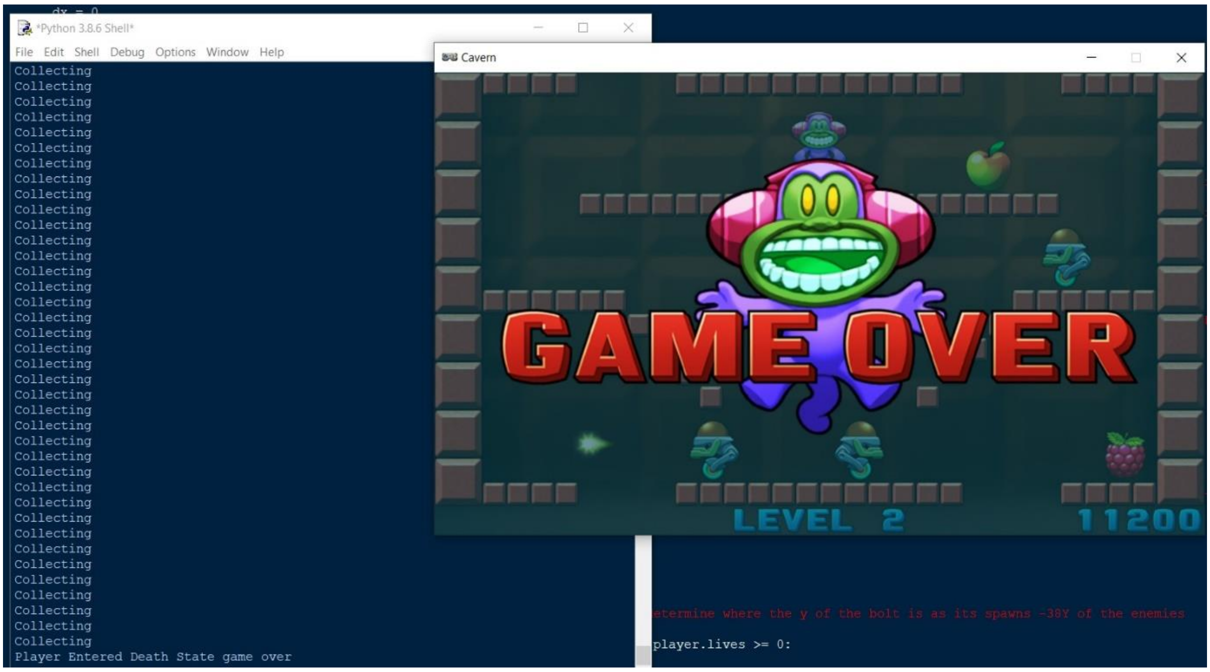
```
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
Player Entered Death State is not dead game over
```

Here you can see that our player has entered the death state (Health =0) notice a problem with this the game has not ended our player is just sitting there.



```
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
Player Entered Death State game over
```

Here you can see again no life’s still the game goes on.

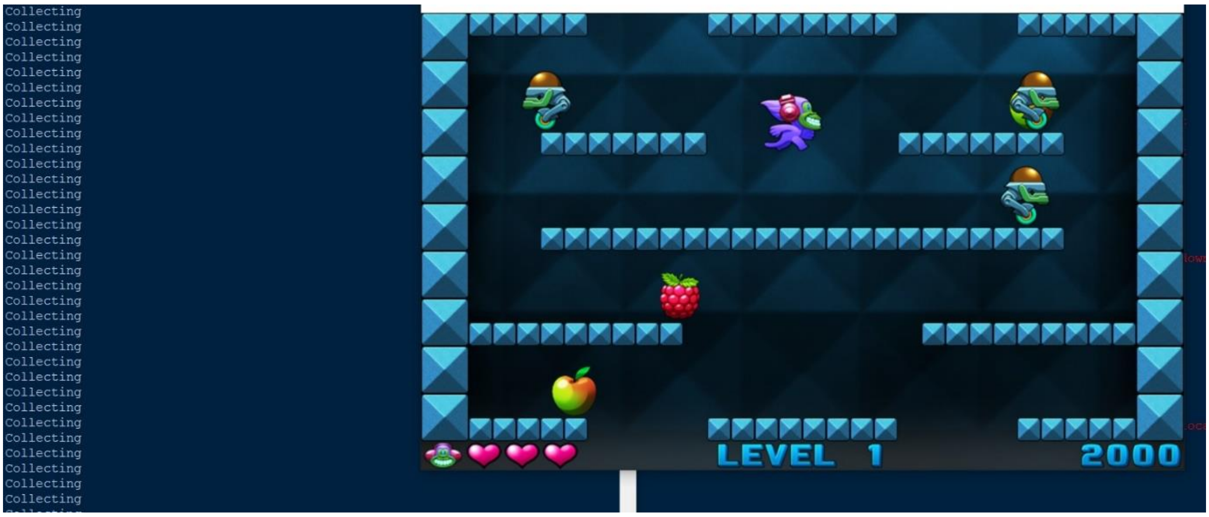


```
dx = 0
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Player Entered Death State game over
```

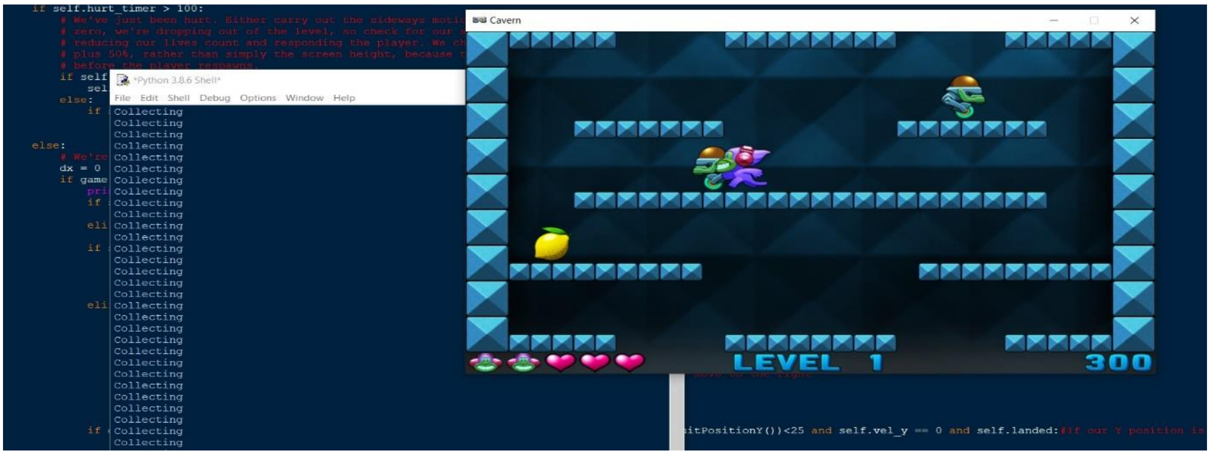
```
etermine where the y of the bolt is as its spawns -38Y of the enemies
player.lives >= 0:
```

But here you see hooray it has entered the death state and entered the game over screen.

Collect:



```
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
```

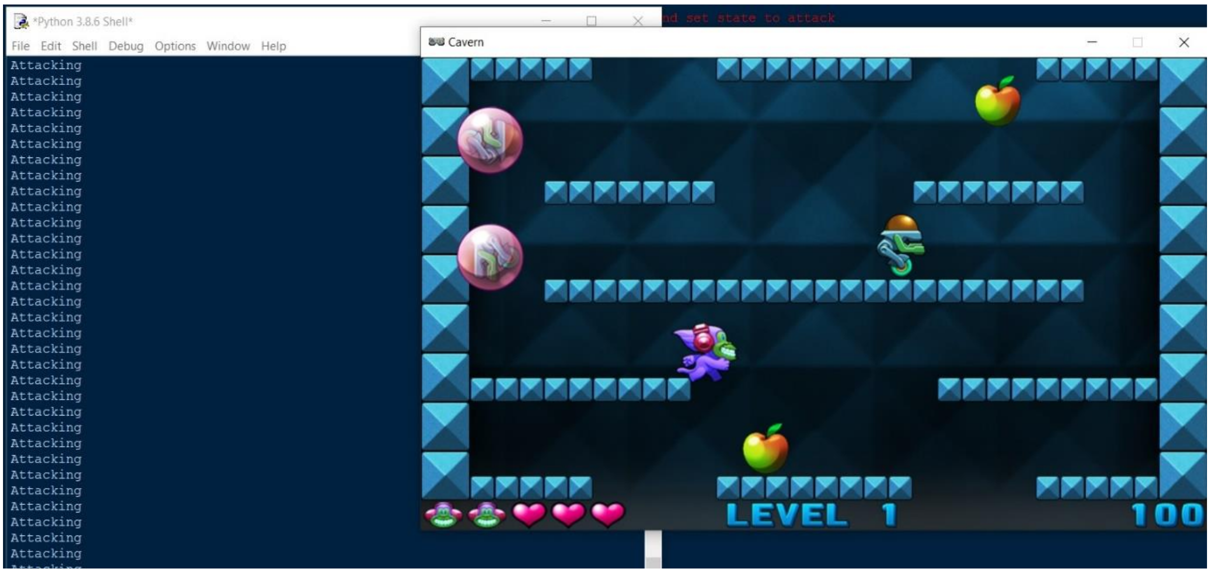


```
    if self.hurt_timer > 100:
        # We've just been hurt. Either carry out the sideways motio
        # zero, we're dropping out of the level, so check for our s
        # reducing our lives count and responding the player. We ch
        # plus 50%, rather than simply the screen height, because t
        # before the player respawns.
        if self
            sel
        else:
            if
    else:
        # We're
        dx = 0
        if game
            pri
            if
            eli
            if
            eli
            if
itPositionY()<25 and self.vel_y == 0 and self.landed:#if our Y position is
```

```
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
```

Here you can see the testing of the collection state our ai is heading to the directions of the fruit effectively collecting them while collecting is printed on the screen that means we are in the collection state.

Attack:

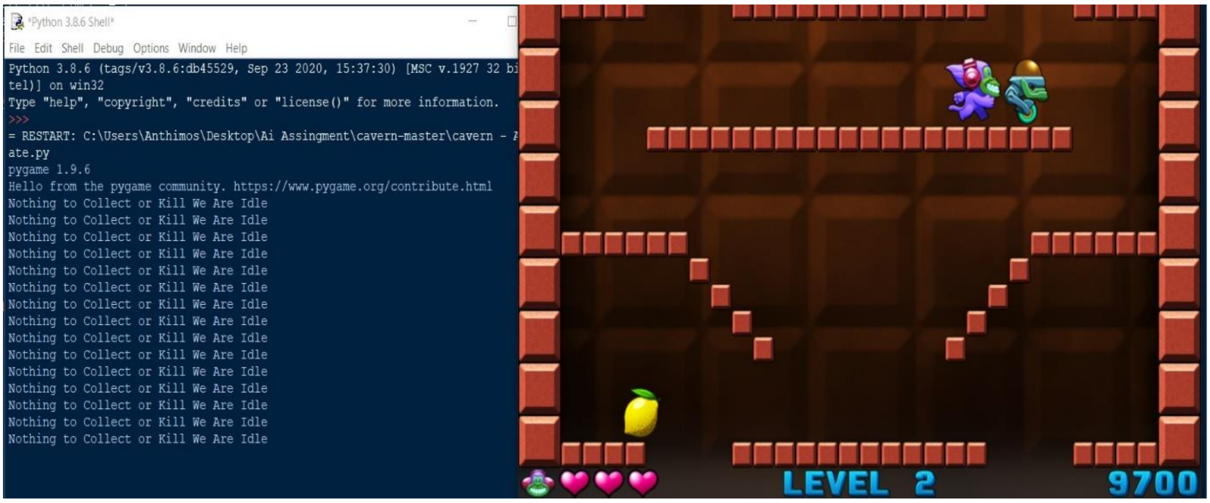


```
nd set state to attack
```

```
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
Attacking
```

In the above image you can see our player has entered its attack state and attacked 2 enemies so we know our attack stage works.

IDLE:

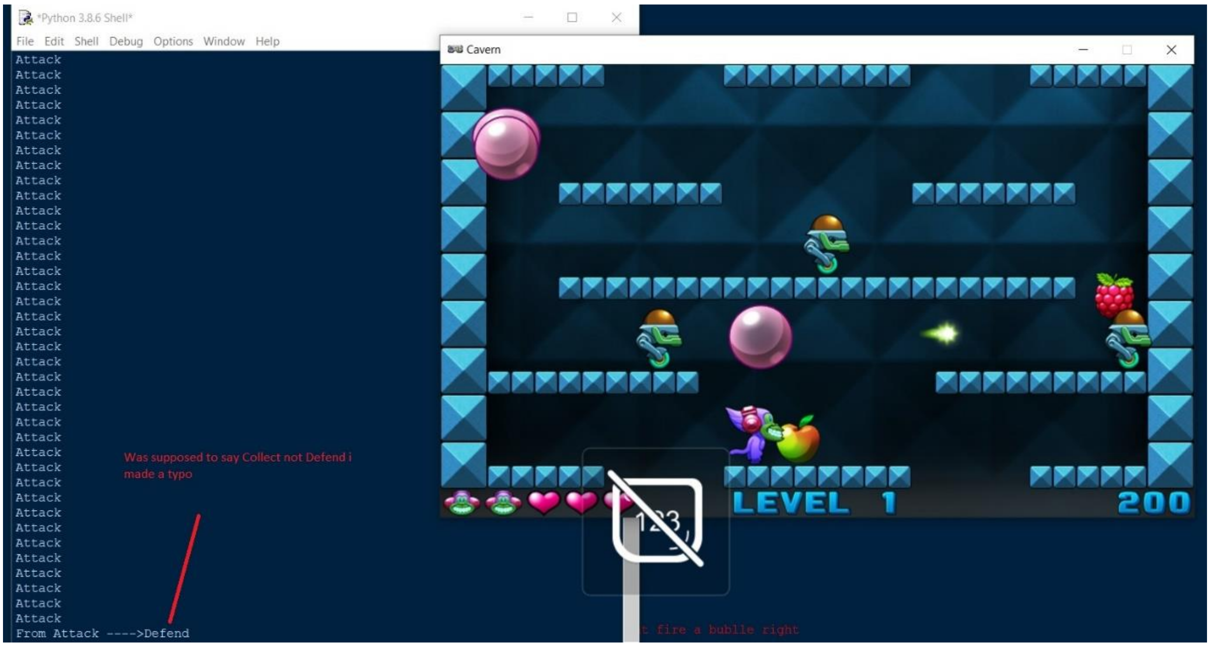


```
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:37:30) [MSC v.1927 32 bi
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Anthimos\Desktop\Ai Assingment\cavern-master\cavern - A
ate.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
Nothing to Collect or Kill We Are Idle
```

This was the test run for the idle state to check if it works you can see on the printed side of the screen we were in the idle state for a couple seconds until level 2 loaded up the idle state is there to prevent out of range errors as explained above the reason why the player moves in the picture is because a fruit and enemy spawned so its state changed.

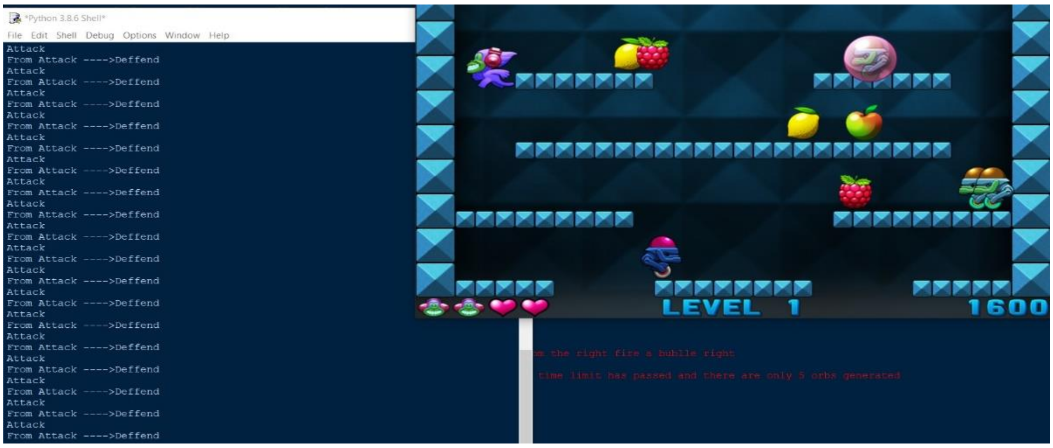
Transitions Tests:

On this image you can see the transition from attack to defend state the code was set up to just transition if a bolt was fired for the sake of checking the transition works correctly.



```
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
From Attack ---->Defend
```

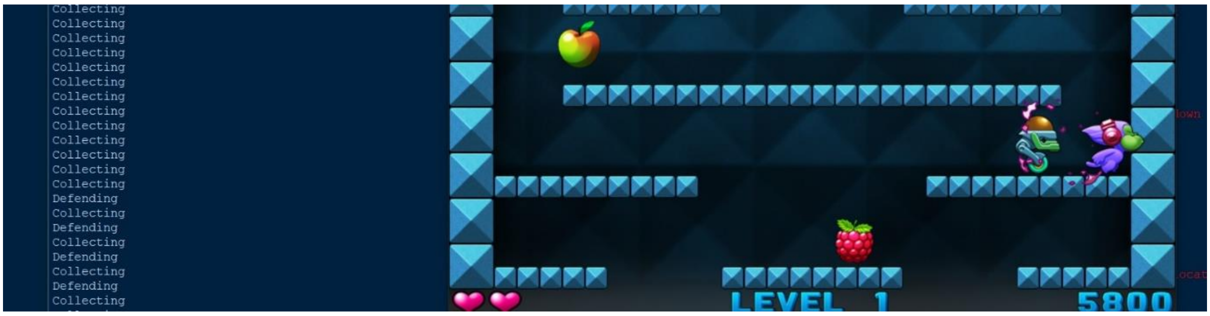
Was supposed to say Collect not Defend i made a typo



```
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
Attack
From Attack ---->Deffend
```

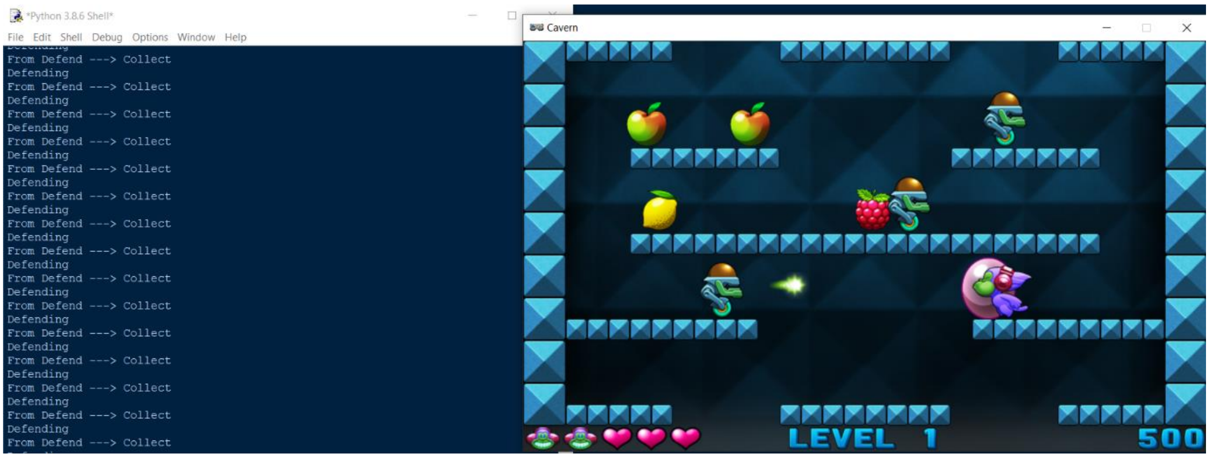
```
m the right fire a bubble right
 time limit has passed and there are only 5 orbs generated
```

On this image you can see the test run for changing from collection state to defending state.



```
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Collecting
Defending
Collecting
Defending
Collecting
Defending
Collecting
Defending
Collecting
```

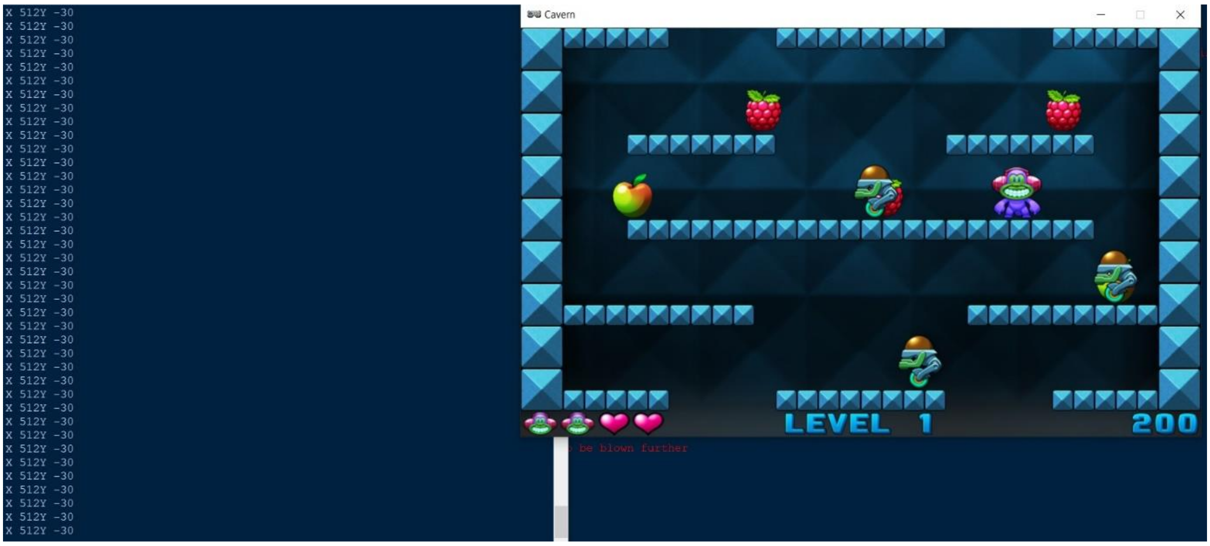
Testing that we return to collection state after we fire an orb to defend our self's.



```
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
```

General Tests:

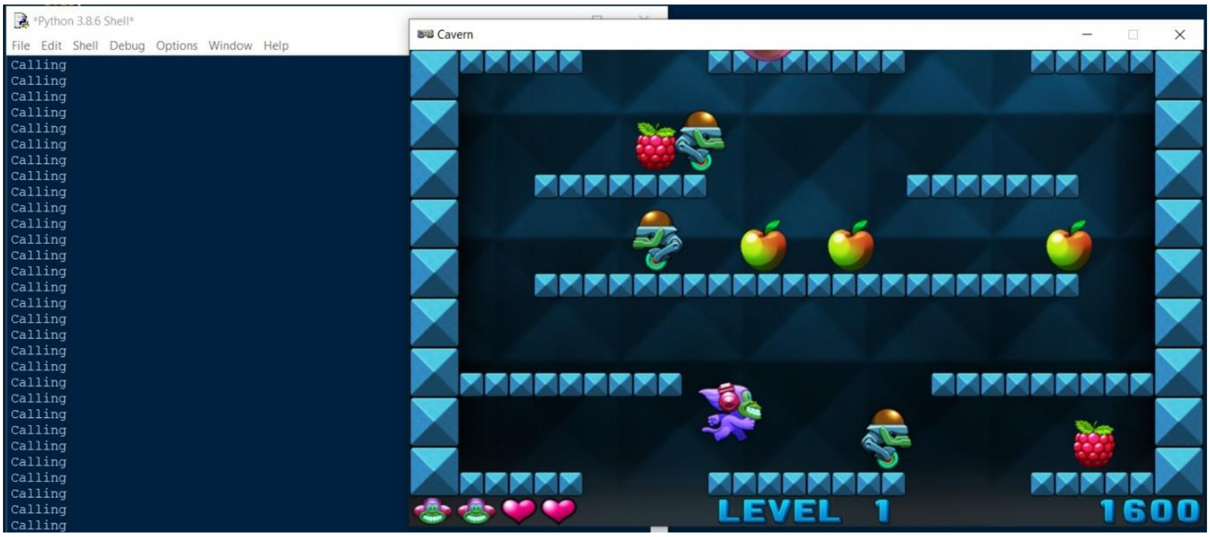
Used to understand how robots' position is logged:



```
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
X 512Y -30
```

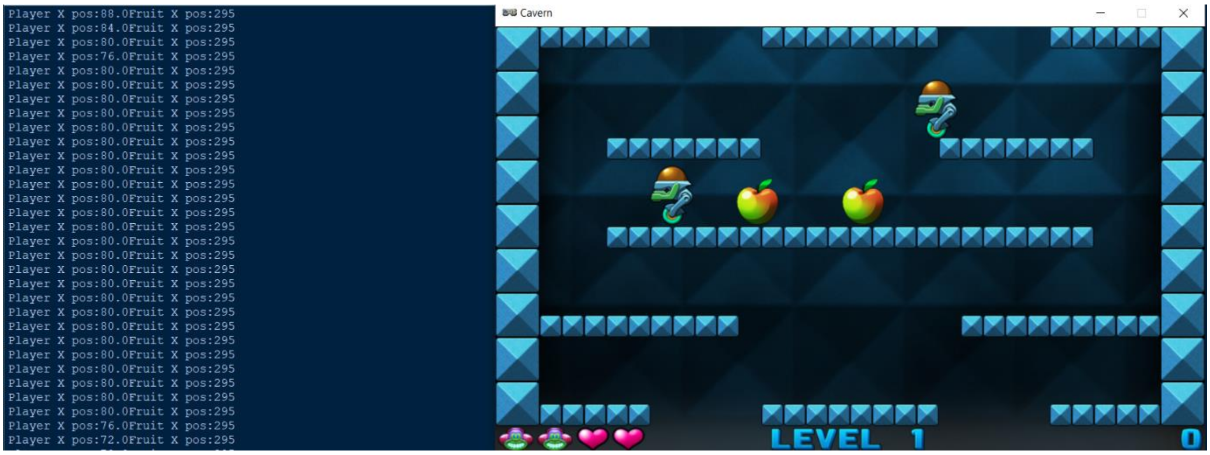
```
be blown further
```

When Calling was printed that meant that we collided with a boundary and changed direction:

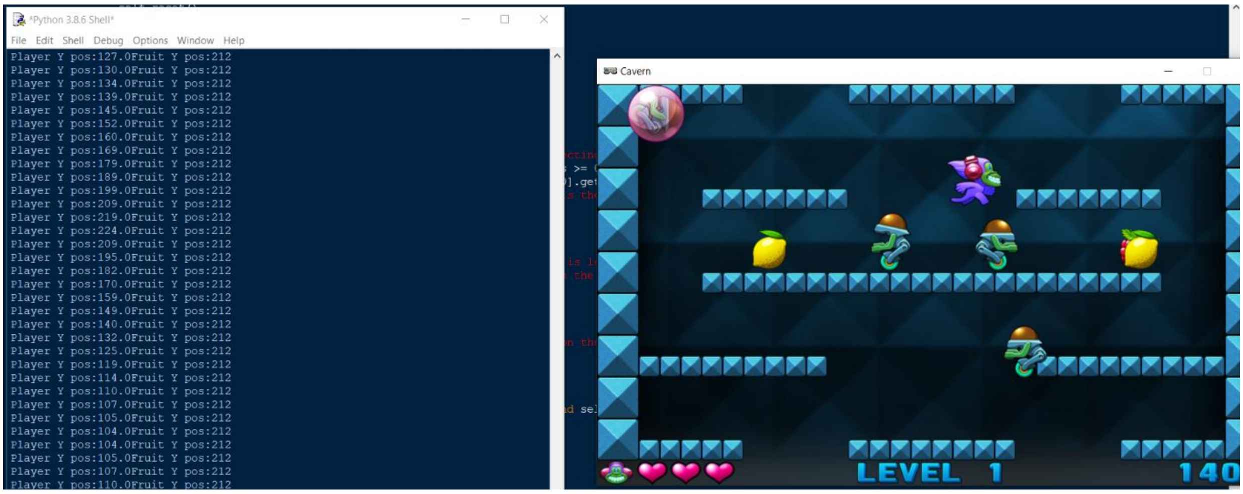


```
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
Calling
```

Check that the player position is changing when going after the fruit also used fruit as reference:

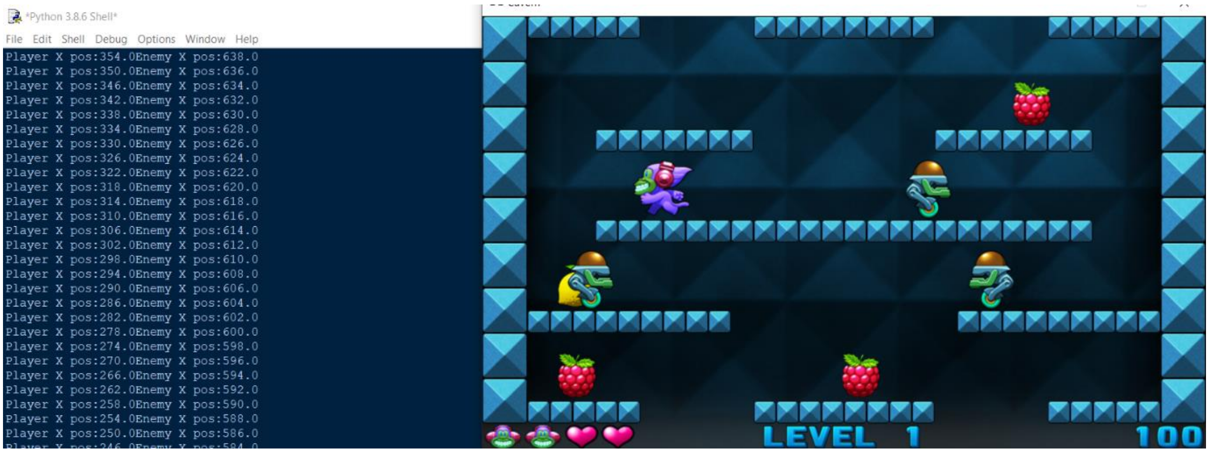


```
Player X pos:88.0Fruit X pos:295
Player X pos:84.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:76.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:80.0Fruit X pos:295
Player X pos:76.0Fruit X pos:295
Player X pos:72.0Fruit X pos:295
```

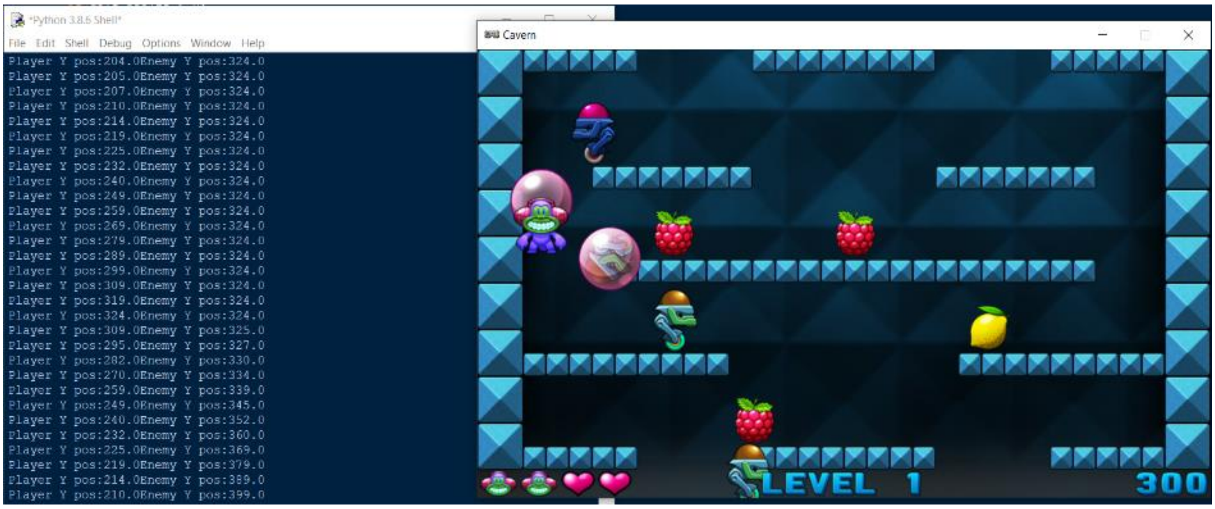


```
Player Y pos:127.0Fruit Y pos:212
Player Y pos:130.0Fruit Y pos:212
Player Y pos:134.0Fruit Y pos:212
Player Y pos:139.0Fruit Y pos:212
Player Y pos:145.0Fruit Y pos:212
Player Y pos:152.0Fruit Y pos:212
Player Y pos:160.0Fruit Y pos:212
Player Y pos:169.0Fruit Y pos:212
Player Y pos:179.0Fruit Y pos:212
Player Y pos:189.0Fruit Y pos:212
Player Y pos:199.0Fruit Y pos:212
Player Y pos:209.0Fruit Y pos:212
Player Y pos:219.0Fruit Y pos:212
Player Y pos:224.0Fruit Y pos:212
Player Y pos:209.0Fruit Y pos:212
Player Y pos:195.0Fruit Y pos:212
Player Y pos:182.0Fruit Y pos:212
Player Y pos:170.0Fruit Y pos:212
Player Y pos:159.0Fruit Y pos:212
Player Y pos:149.0Fruit Y pos:212
Player Y pos:140.0Fruit Y pos:212
Player Y pos:132.0Fruit Y pos:212
Player Y pos:125.0Fruit Y pos:212
Player Y pos:119.0Fruit Y pos:212
Player Y pos:114.0Fruit Y pos:212
Player Y pos:110.0Fruit Y pos:212
Player Y pos:107.0Fruit Y pos:212
Player Y pos:105.0Fruit Y pos:212
Player Y pos:104.0Fruit Y pos:212
Player Y pos:104.0Fruit Y pos:212
Player Y pos:105.0Fruit Y pos:212
Player Y pos:107.0Fruit Y pos:212
Player Y pos:110.0Fruit Y pos:212
```

Checking enemies and Player Positions:

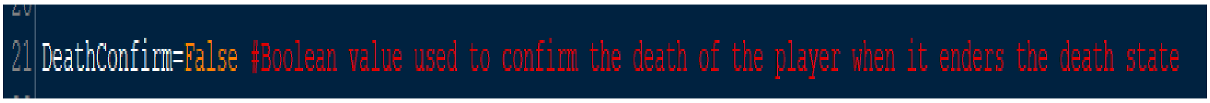


```
Player X pos:354.0Enemy X pos:638.0
Player X pos:350.0Enemy X pos:636.0
Player X pos:346.0Enemy X pos:634.0
Player X pos:342.0Enemy X pos:632.0
Player X pos:338.0Enemy X pos:630.0
Player X pos:334.0Enemy X pos:628.0
Player X pos:330.0Enemy X pos:626.0
Player X pos:326.0Enemy X pos:624.0
Player X pos:322.0Enemy X pos:622.0
Player X pos:318.0Enemy X pos:620.0
Player X pos:314.0Enemy X pos:618.0
Player X pos:310.0Enemy X pos:616.0
Player X pos:306.0Enemy X pos:614.0
Player X pos:302.0Enemy X pos:612.0
Player X pos:298.0Enemy X pos:610.0
Player X pos:294.0Enemy X pos:608.0
Player X pos:290.0Enemy X pos:606.0
Player X pos:286.0Enemy X pos:604.0
Player X pos:282.0Enemy X pos:602.0
Player X pos:278.0Enemy X pos:600.0
Player X pos:274.0Enemy X pos:598.0
Player X pos:270.0Enemy X pos:596.0
Player X pos:266.0Enemy X pos:594.0
Player X pos:262.0Enemy X pos:592.0
Player X pos:258.0Enemy X pos:590.0
Player X pos:254.0Enemy X pos:588.0
Player X pos:250.0Enemy X pos:586.0
Player X pos:246.0Enemy X pos:584.0
```



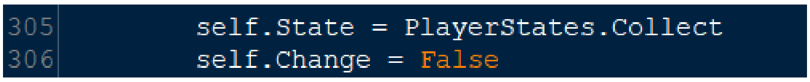
```
Player Y pos:204.0Enemy Y pos:324.0
Player Y pos:205.0Enemy Y pos:324.0
Player Y pos:207.0Enemy Y pos:324.0
Player Y pos:210.0Enemy Y pos:324.0
Player Y pos:214.0Enemy Y pos:324.0
Player Y pos:219.0Enemy Y pos:324.0
Player Y pos:225.0Enemy Y pos:324.0
Player Y pos:232.0Enemy Y pos:324.0
Player Y pos:240.0Enemy Y pos:324.0
Player Y pos:249.0Enemy Y pos:324.0
Player Y pos:259.0Enemy Y pos:324.0
Player Y pos:269.0Enemy Y pos:324.0
Player Y pos:279.0Enemy Y pos:324.0
Player Y pos:289.0Enemy Y pos:324.0
Player Y pos:299.0Enemy Y pos:324.0
Player Y pos:309.0Enemy Y pos:324.0
Player Y pos:319.0Enemy Y pos:324.0
Player Y pos:324.0Enemy Y pos:324.0
Player Y pos:309.0Enemy Y pos:325.0
Player Y pos:295.0Enemy Y pos:327.0
Player Y pos:282.0Enemy Y pos:330.0
Player Y pos:270.0Enemy Y pos:334.0
Player Y pos:259.0Enemy Y pos:339.0
Player Y pos:249.0Enemy Y pos:345.0
Player Y pos:240.0Enemy Y pos:352.0
Player Y pos:232.0Enemy Y pos:360.0
Player Y pos:225.0Enemy Y pos:369.0
Player Y pos:219.0Enemy Y pos:379.0
Player Y pos:214.0Enemy Y pos:389.0
Player Y pos:210.0Enemy Y pos:399.0
```

Code Snippets And Explanation:



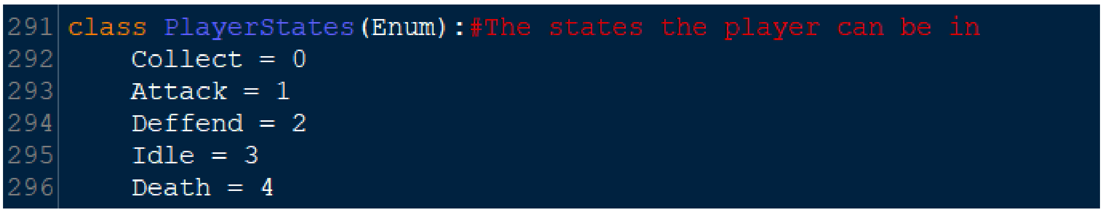
```
DeathConfirm=False #Boolean value used to confirm the death of the player when it enders the death state
```

The confirm death Boolean is used to check that our player has died it is a global and its passed in the update function of the player class and the main one of the game it is set true when we enter our death state if its true the game ends.



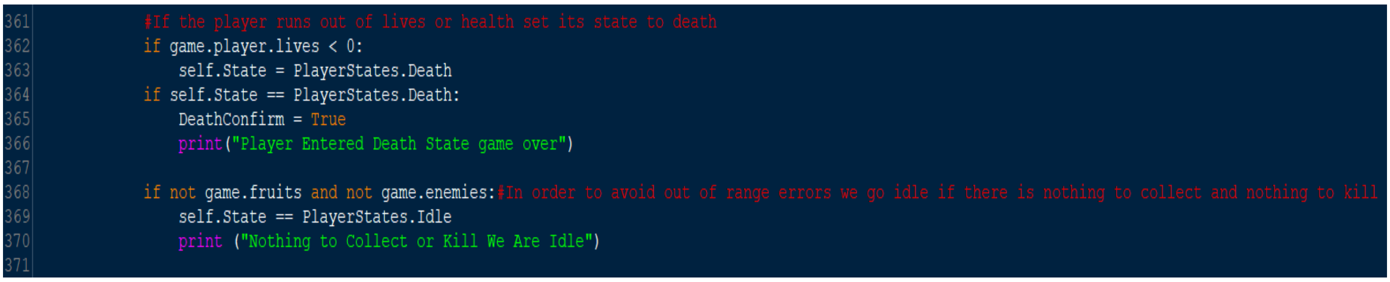
```
        self.State = PlayerStates.Collect
        self.Change = False
```

This two self-variables are set in the player class in its init function the change is a Boolean used to change the direction of the player if it collides with a wall the state is used to keep the state we are in.



```
class PlayerStates(Enum):#The states the player can be in
    Collect = 0
    Attack = 1
    Deffend = 2
    Idle = 3
    Death = 4
```

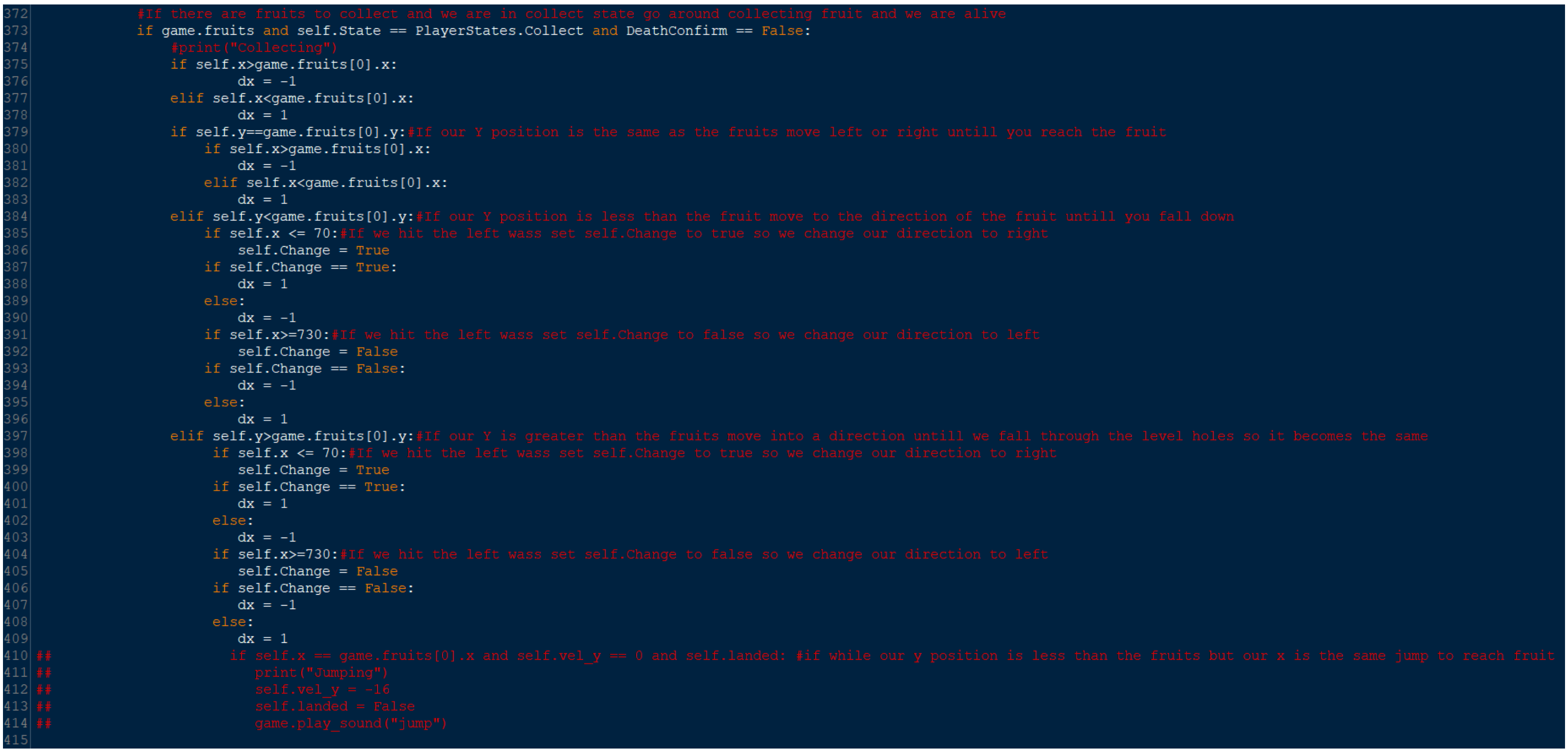
The player states class is set right before the player class its of type Enum and it holds the states that our player can be in.



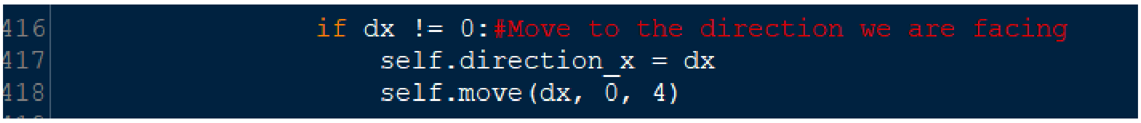
```
        #If the player runs out of lives or health set its state to death
        if game.player.lives < 0:
            self.State = PlayerStates.Death
        if self.State == PlayerStates.Death:
            DeathConfirm = True
            print("Player Entered Death State game over")

        if not game.fruits and not game.enemies:#In order to avoid out of range errors we go idle if there is nothing to collect and nothing to kill
            self.State == PlayerStates.Idle
            print ("Nothing to Collect or Kill We Are Idle")
```

This 3 if statements are set in our players update function where the handling of the movement was done with the keyboard, the first one checks if our player objects lives are bellow 0 and sets the players state to Death the next one checks if our state is Death and Sets the DeathConfirm bool to true in order to end the game.

The third one check’s if there are no fruit and no enemies in the game and sets our state to Idle to avoid our states from working and breaking the game with out of range errors.



```
            #If there are fruits to collect and we are in collect state go around collecting fruit and we are alive
            if game.fruits and self.State == PlayerStates.Collect and DeathConfirm == False:
                #print("Collecting")
                if self.x>game.fruits[0].x:
                        dx = -1
                elif self.x<game.fruits[0].x:
                        dx = 1
                if self.y==game.fruits[0].y:#If our Y position is the same as the fruits move left or right untill you reach the fruit
                    if self.x>game.fruits[0].x:
                        dx = -1
                    elif self.x<game.fruits[0].x:
                        dx = 1
                elif self.y<game.fruits[0].y:#If our Y position is less than the fruit move to the direction of the fruit untill you fall down
                    if self.x <= 70:#If we hit the left wass set self.Change to true so we change our direction to right
                        self.Change = True
                    if self.Change == True:
                        dx = 1
                    else:
                        dx = -1
                    if self.x>=730:#If we hit the left wass set self.Change to false so we change our direction to left
                        self.Change = False
                    if self.Change == False:
                        dx = -1
                    else:
                        dx = 1
                elif self.y>game.fruits[0].y:#If our Y is greater than the fruits move into a direction untill we fall through the level holes so it becomes the same
                     if self.x <= 70:#If we hit the left wass set self.Change to true so we change our direction to right
                        self.Change = True
                     if self.Change == True:
                        dx = 1
                     else:
                        dx = -1
                     if self.x>=730:#If we hit the left wass set self.Change to false so we change our direction to left
                        self.Change = False
                     if self.Change == False:
                        dx = -1
                     else:
                        dx = 1
##                     if self.x == game.fruits[0].x and self.vel_y == 0 and self.landed: #if while our y position is less than the fruits but our x is the same jump to reach fruit
##                        print("Jumping")
##                        self.vel_y = -16
##                        self.landed = False
##                        game.play_sound("jump")

```

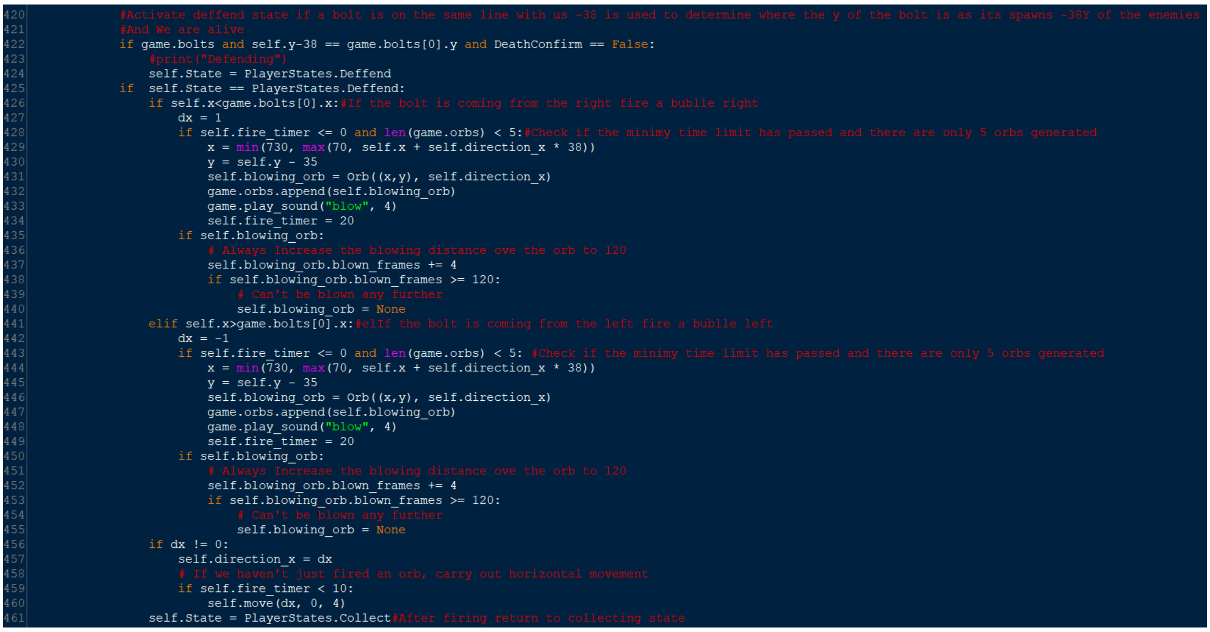


```
if dx != 0:#Move to the direction we are facing
    self.direction_x = dx
    self.move(dx, 0, 4)
```

This is our collection state code we are by default set in the collection state but in order for us to do something there needs to be fruits in the game our state to be correct and we are not dead.

The first 2 if statements will check where our fruit location is and set our direction accordingly either left or right then we check if we are on the same Y with the fruit we will move again accordingly on the x to take the fruit if we our Y is less than the fruit we will move left until we hit the left wall if we hit the wall we will change our direction to the right if our Y is greater than the fruit we will move to the right until we hit the right wall if that happens we will change direction too the left you will notice that there is a commented out function that is our jumping function jumping was implemented in early stages of the code in order to move from platform to platform and grab the fruit as that will cause issues a simpler approach was implemented that made use of the games dropping down from the end to the top in order to go to the y position desired.

Last our direction if statement was changed slightly so it does not check if an orb was fired as that is done in the defend class will just keep moving to the desired direction.



```
        #Activate deffend state if a bolt is on the same line with us -38 is used to determine where the y of the bolt is as its spawns -38Y of the enemies
        #And We are alive
        if game.bolts and self.y-38 == game.bolts[0].y and DeathConfirm == False:
            #print("Defending")
            self.State = PlayerStates.Deffend
        if  self.State == PlayerStates.Deffend:
            if self.x<game.bolts[0].x:#If the bolt is coming from the right fire a bublle right
                dx = 1
                if self.fire_timer <= 0 and len(game.orbs) < 5:#Check if the minimy time limit has passed and there are only 5 orbs generated
                    x = min(730, max(70, self.x + self.direction_x * 38))
                    y = self.y - 35
                    self.blowing_orb = Orb((x,y), self.direction_x)
                    game.orbs.append(self.blowing_orb)
                    game.play_sound("blow", 4)
                    self.fire_timer = 20
                if self.blowing_orb:
                    # Always Increase the blowing distance ove the orb to 120
                    self.blowing_orb.blown_frames += 4
                    if self.blowing_orb.blown_frames >= 120:
                        # Can't be blown any further
                        self.blowing_orb = None
            elif self.x>game.bolts[0].x:#elIf the bolt is coming from the left fire a bublle left
                dx = -1
                if self.fire_timer <= 0 and len(game.orbs) < 5: #Check if the minimy time limit has passed and there are only 5 orbs generated
                    x = min(730, max(70, self.x + self.direction_x * 38))
                    y = self.y - 35
                    self.blowing_orb = Orb((x,y), self.direction_x)
                    game.orbs.append(self.blowing_orb)
                    game.play_sound("blow", 4)
                    self.fire_timer = 20
                if self.blowing_orb:
                    # Always Increase the blowing distance ove the orb to 120
                    self.blowing_orb.blown_frames += 4
                    if self.blowing_orb.blown_frames >= 120:
                        # Can't be blown any further
                        self.blowing_orb = None
            if dx != 0:
                self.direction_x = dx
                # If we haven't just fired an orb, carry out horizontal movement
                if self.fire_timer < 10:
                    self.move(dx, 0, 4)
            self.State = PlayerStates.Collect#After firing return to collecting state
```

This is our Defend state its only activated when there bolts in the game and that bolt has the same Y position with us.

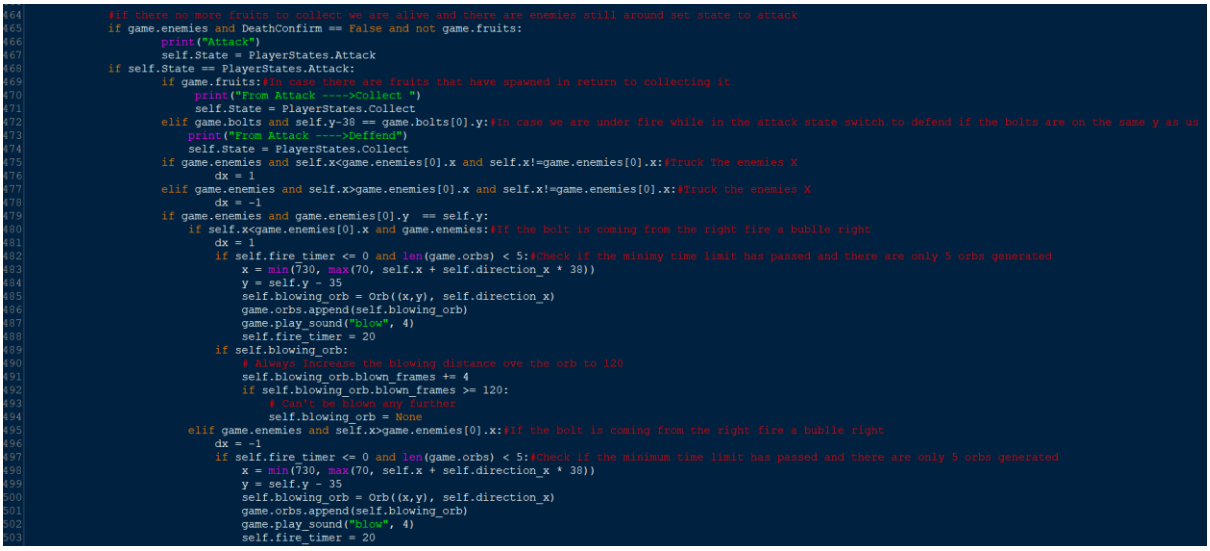
It makes use of the already existing code for firing orbs the only difference it checks the direction of the bolt if the player position is less than the bolts one it means it’s coming from the right so we turn right and fire our orbs only if our fire timer is less than or equal to 0 and we cant fire more than 5 orbs at a time.

If our position is greater than the bolts it means its coming from the left, so we turn left and repeat the process.

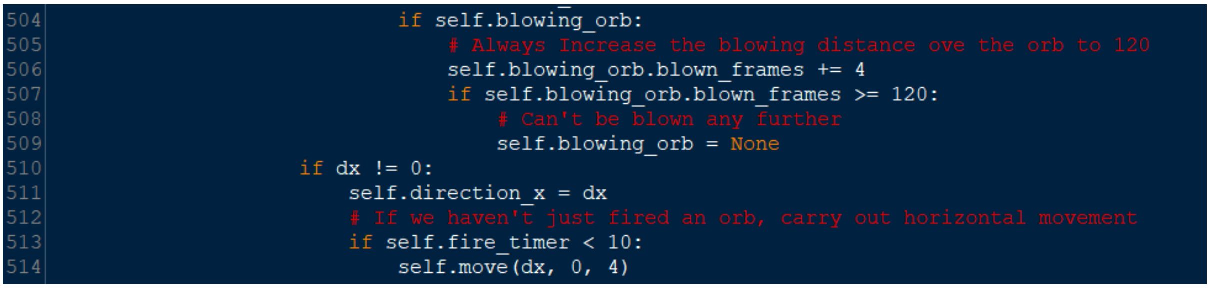
The blowing distance of the orb will always be increased for maximum effect.

The direction function stays the same as the game one because we are firing orbs.

After we fire, we return to the collection state.



```
        #if there no more fruits to collect we are alive and there are enemies still around set state to attack
        if game.enemies and DeathConfirm == False and not game.fruits:
                print("Attack")
                self.State = PlayerStates.Attack
        if self.State == PlayerStates.Attack:
                if game.fruits:#In case there are fruits that have spawned in return to collecting it
                     print("From Attack ---->Collect ")
                     self.State = PlayerStates.Collect
                elif game.bolts and self.y-38 == game.bolts[0].y:#In case we are under fire while in the attack state switch to defend if the bolts are on the same y as us
                    print("From Attack ---->Deffend")
                    self.State = PlayerStates.Collect
                if game.enemies and self.x<game.enemies[0].x and self.x!=game.enemies[0].x:#Truck The enemies X
                        dx = 1
                elif game.enemies and self.x>game.enemies[0].x and self.x!=game.enemies[0].x:#Truck the enemies X
                        dx = -1
                if game.enemies and game.enemies[0].y  == self.y:
                    if self.x<game.enemies[0].x and game.enemies:#If the bolt is coming from the right fire a bublle right
                        dx = 1
                        if self.fire_timer <= 0 and len(game.orbs) < 5:#Check if the minimy time limit has passed and there are only 5 orbs generated
                            x = min(730, max(70, self.x + self.direction_x * 38))
                            y = self.y - 35
                            self.blowing_orb = Orb((x,y), self.direction_x)
                            game.orbs.append(self.blowing_orb)
                            game.play_sound("blow", 4)
                            self.fire_timer = 20
                        if self.blowing_orb:
                            # Always Increase the blowing distance ove the orb to 120
                            self.blowing_orb.blown_frames += 4
                            if self.blowing_orb.blown_frames >= 120:
                                # Can't be blown any further
                                self.blowing_orb = None
                    elif game.enemies and self.x>game.enemies[0].x:#If the bolt is coming from the right fire a bublle right
                        dx = -1
                        if self.fire_timer <= 0 and len(game.orbs) < 5:#Check if the minimum time limit has passed and there are only 5 orbs generated
                            x = min(730, max(70, self.x + self.direction_x * 38))
                            y = self.y - 35
                            self.blowing_orb = Orb((x,y), self.direction_x)
                            game.orbs.append(self.blowing_orb)
                            game.play_sound("blow", 4)
                            self.fire_timer = 20
```



```
                            if self.blowing_orb:
                                # Always Increase the blowing distance ove the orb to 120
                                self.blowing_orb.blown_frames += 4
                                if self.blowing_orb.blown_frames >= 120:
                                    # Can't be blown any further
                                    self.blowing_orb = None
                    if dx != 0:
                        self.direction_x = dx
                        # If we haven't just fired an orb, carry out horizontal movement
                        if self.fire_timer < 10:
                            self.move(dx, 0, 4)
```

This is our attack state we will only enter it if there are enemies and no fruit.

In the case while we are in the attack state and fruit spawn in, we will move back to our collection state.

In the case we are in the attack state and a bolt fires at us will transition to the defend state.

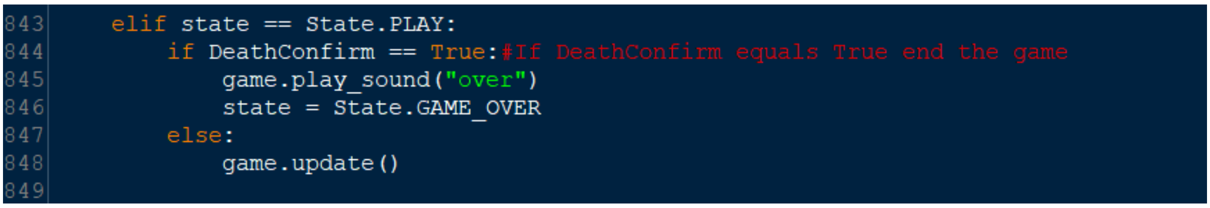
This is handled by our if and else if statements in lines 469-474.

If nothing of the above applies we will continue as normal.

The firing of the orbs code and the direction statement was not changed as we need it for our orbs to work as expected.

With our second if else if statements we truck the enemies x position moving the way the enemy is moving.

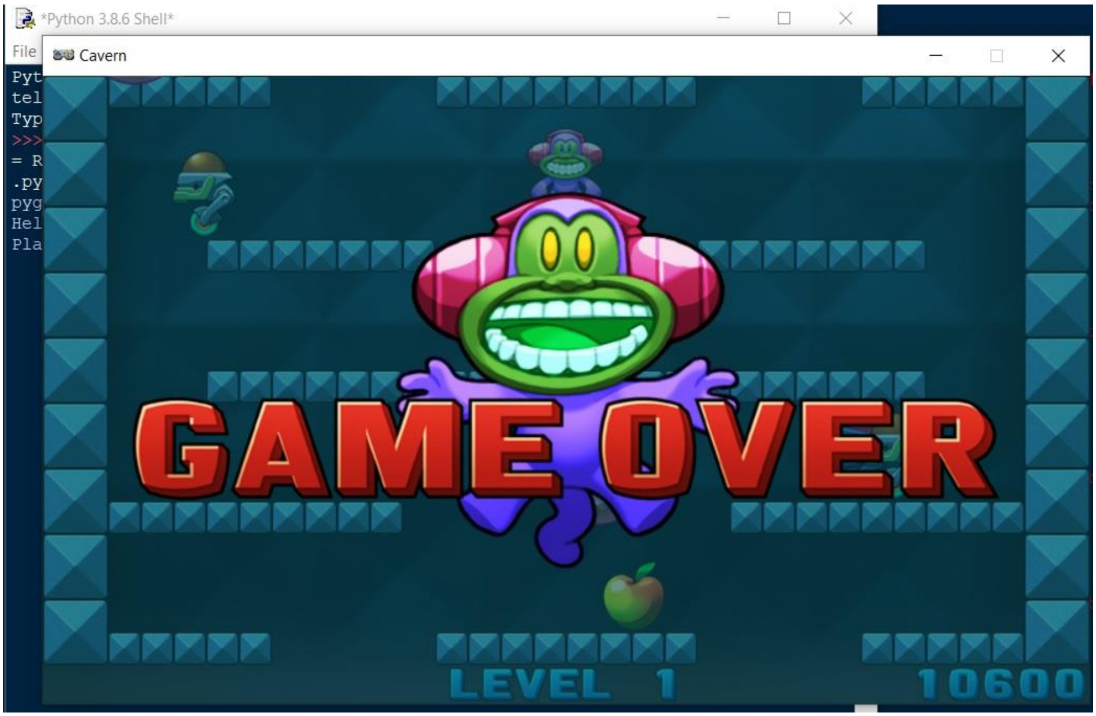
Then if our enemies y position is the same as the player, we check our x position if its less or greater than the enemies and fire orbs at the appropriate direction.



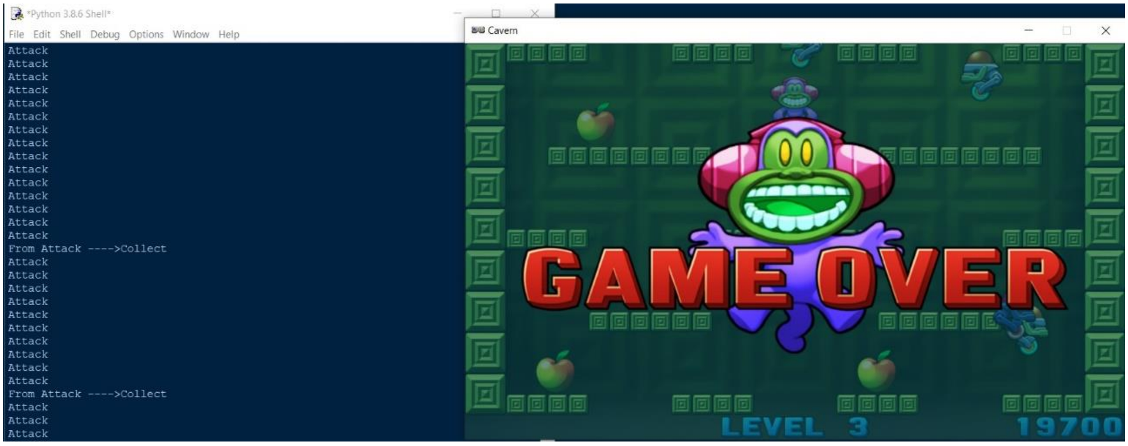
```
    elif state == State.PLAY:
        if DeathConfirm == True:#If DeathConfirm equals True end the game
            game.play_sound("over")
            state = State.GAME_OVER
        else:
            game.update()

```

The last thing changed was the if statement in the main update function of the game it just checks that DeathConfirm is true if it is that means we are in our death state so end the game.

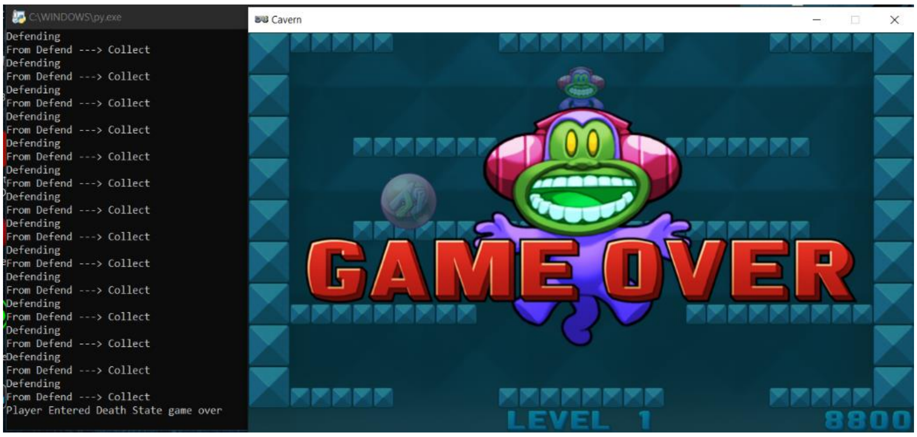
Result Tests:



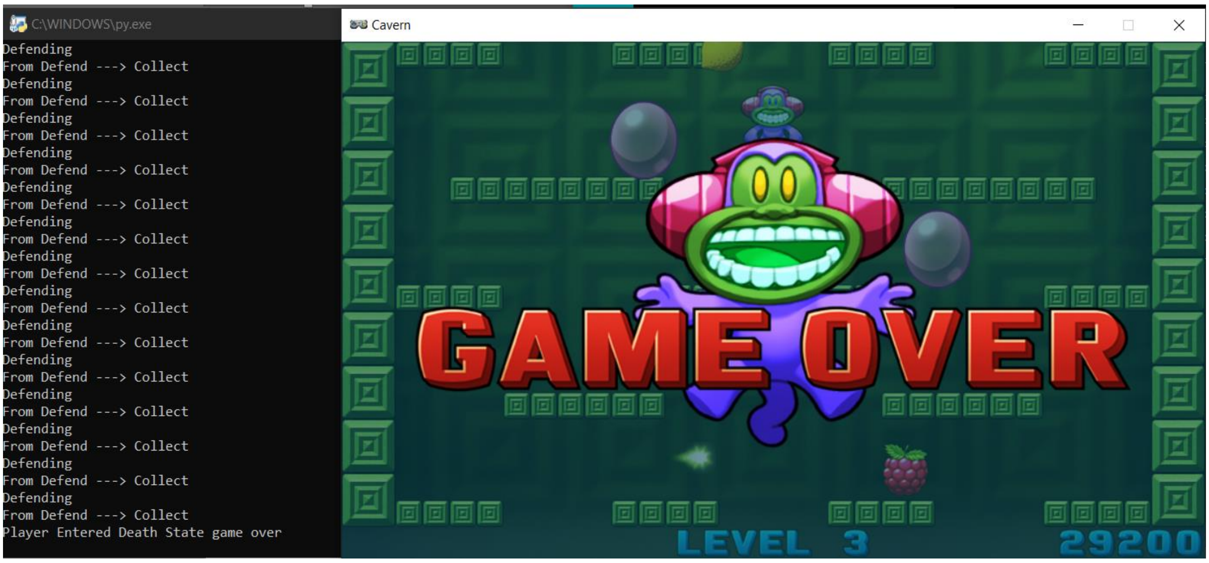
```
Pyt
tel
Typ
>>>
= R
.py
pyg
Hel
Pla
```



```
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
From Attack ---->Collect
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
Attack
From Attack ---->Collect
Attack
Attack
Attack
```



```
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Player Entered Death State game over
```



```
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Defending
From Defend ---> Collect
Player Entered Death State game over
```

Bugs:

1) Sometimes when trying to defend and we are facing the right direction the orbs my spawn behind us (Might be in my mind).
2) The jumping will cause problems with the direction: Jumping was disabled (Fixed)
3) The player will get stuck on walls: Fixed
4) On level 2 sometimes the player might get stuck on a corner of the stairs
5) Out of index errors will be thrown even if we implemented idle state: Fixed by forcing all the if statements to check that are actually enemies because the code had to go through them if an enemy was left didn't know what to do if it was killed.
6) When entering death state, the game would not end: Fixed with the use of our Boolean
7) Player would not know what to do if its Y position was less than the fruit, but its X was the same: Fixed with the introduction of our if statements above.