imagine that you have built a library and that I am a client that wants to use this library to train DQN on objects from games (not only assault). Your goal is to present how your library works to me and to explain what I can do with it.

Because now, we don’t know for example, if I put "vision" or "ram" what will be the differences in the observation shape. What if I put "revised"? Another example, what if on a frame there are 8 objects and on the next frame there are 9, then how do you specify the global observation shape size? Is there a way to get the maximum number of objects that could appear in a frame?

I think you should take the time to really dive deep into the source code. If you do that once and for all, you will save a lot of time for the rest of the project. So let us take a step back from training and RL right now, and focus more on the engineering aspect of the project.

Dear client, if your goal is to train a DQN agent on images, we recommend to use the raw value for the mode argument of the ocatari class. Indeed, when doing so the resulting observation space of the initialized environment will be images, I.E, 3D arrays that are stacked or not. This is done in the source code with the functions ....

##############################################################################  
In ***OC\_Atari\ocatari\core.py:***

The code checks if the torch module (PyTorch), cv2 (OpenCV) and pygame are installed. If any of these is not found, the code suggests installing it with pip install “missing module”.

AVAILABLE\_GAMES = ["Alien", "Assault", "Asterix", "Asteroids", "Atlantis", "BeamRider", "Berzerk", "Bowling", "Boxing",

                   "Breakout", "Carnival", "Centipede", "ChoppperCommand", "DemonAttack", "Enduro", "FishingDerby", "Freeway",

                   "Frostbite", "Gopher", "Hero", "IceHockey", "Kangaroo", "MontezumaRevenge", "MsPacman","Pitfall", "Pong", "PrivateEye",

                   "Qbert", "Riverraid", "RoadRunner", "Seaquest", "Skiing", "SpaceInvaders", "Tennis","Videocube", "Venture", "Yarsrevenge"]

1)Render mode = *human*:   
displays the game in front of you using pygame window, while limiting the fps to 60 to avoid super-fast movement

2)Render mode = *rgb\_array*:   
does not display it, just stores it as a 3d array.

**NB:** Ipdb= IPython Debugger

1)"Vision" Mode:

In "vision" mode, the environment is observed using vision-based sensors, typically capturing the visual information (e.g., pixels, screenshots) of the game screen.

This mode is suitable for tasks that require analyzing and making decisions based on the visual output of the game.

2)"Raw" Mode:

In "raw" mode, the environment is observed using raw data, such as reading the values of specific RAM addresses of the emulator.

This mode is suitable for tasks that do not require visual information and instead rely on specific game state variables stored in RAM.

3)"Revised" Mode:

In "revised" mode, the environment processing is adapted to a specific mode that includes some modifications. This mode is tailored to a particular game environment, and the modifications are likely to be specific to that game.

The detect\_objects\_revised function is used to detect and process objects based on the modified game state variables.

This mode is suitable for games where the default "vision" and "raw" modes do not provide the necessary information, and custom processing is required.

4)"Both" Mode:

In "both" mode, both "vision" and "revised" modes are available simultaneously. This means that the environment can be observed and interacted with using both visual information and modified game state variables.

Separate functions, detect\_objects\_vision and detect\_objects\_revised, are used to detect and process objects based on visual and modified game state data, respectively.

The \_step\_test method is used to interact with the environment and allows you to choose whether to use visual or modified game state information for interaction.

This mode is useful when you want to experiment with agents that utilize both **visual and revised** game state information.

The primary use case for the "both" mode is when you require access to both types of information, likely for debugging, analysis, or specific research purposes. You might want to observe visual information and access game state variables simultaneously to gain insights into how the game environment works or to facilitate specific types of analysis.

***OC\_Atari\scripts\ReverseEngineeringHelper.py***

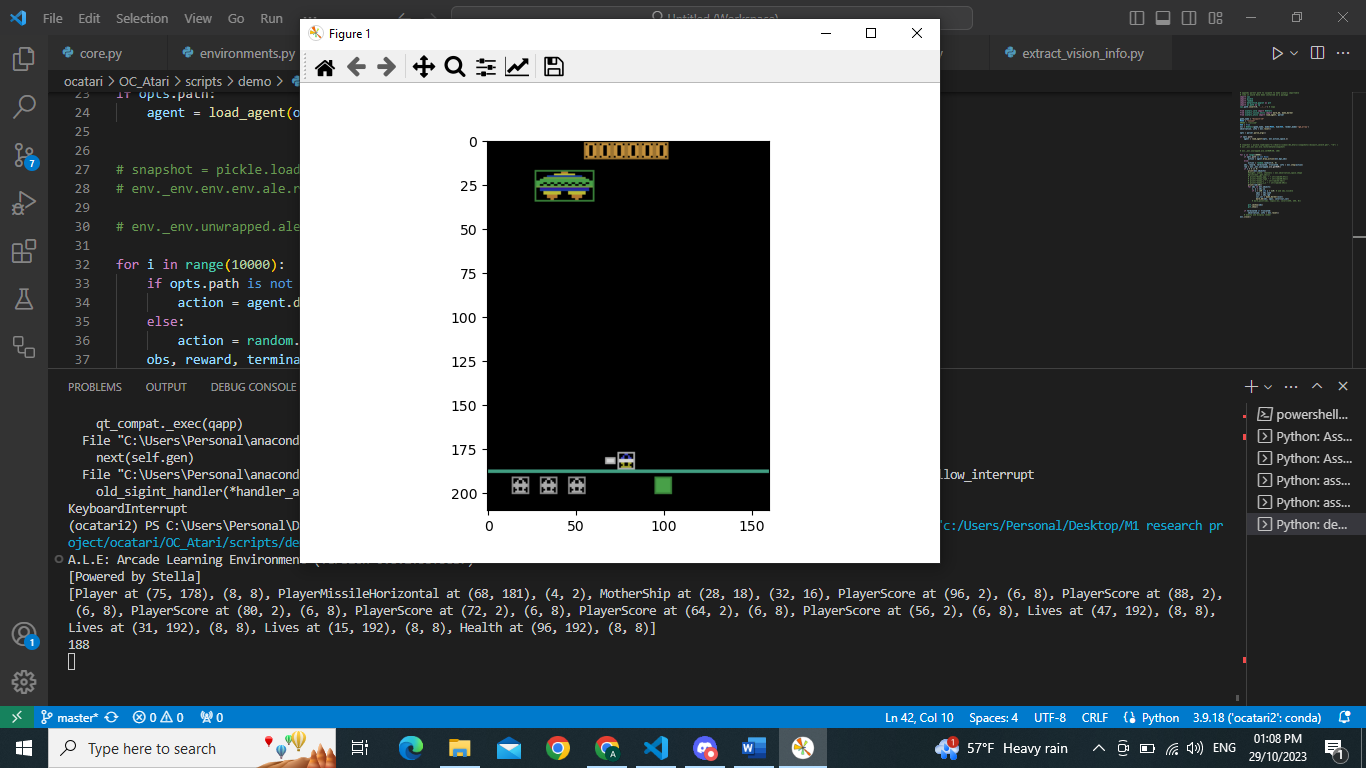
***1)USING “VISION”:***

As we can see here the initial observation was made when we had 13 objects on the screen:

* Player at (75, 178), (8, 8),
* PlayerMissileHorizontal at (68, 181), (4, 2),
* MotherShip at (28, 18), (32, 16),
* PlayerScore at (96, 2), (6, 8), PlayerScore at (88, 2), (6, 8), PlayerScore at (80, 2), (6, 8), PlayerScore at (72, 2), (6, 8), PlayerScore at (64, 2), (6, 8), PlayerScore at (56, 2), (6, 8),
* Lives at (47, 192), (8, 8), Lives at (31, 192), (8, 8), Lives at (15, 192), (8, 8),
* Health at (96, 192), (8, 8)

Notice that first 2 digits being the coordinates, the second two being their dimensions.

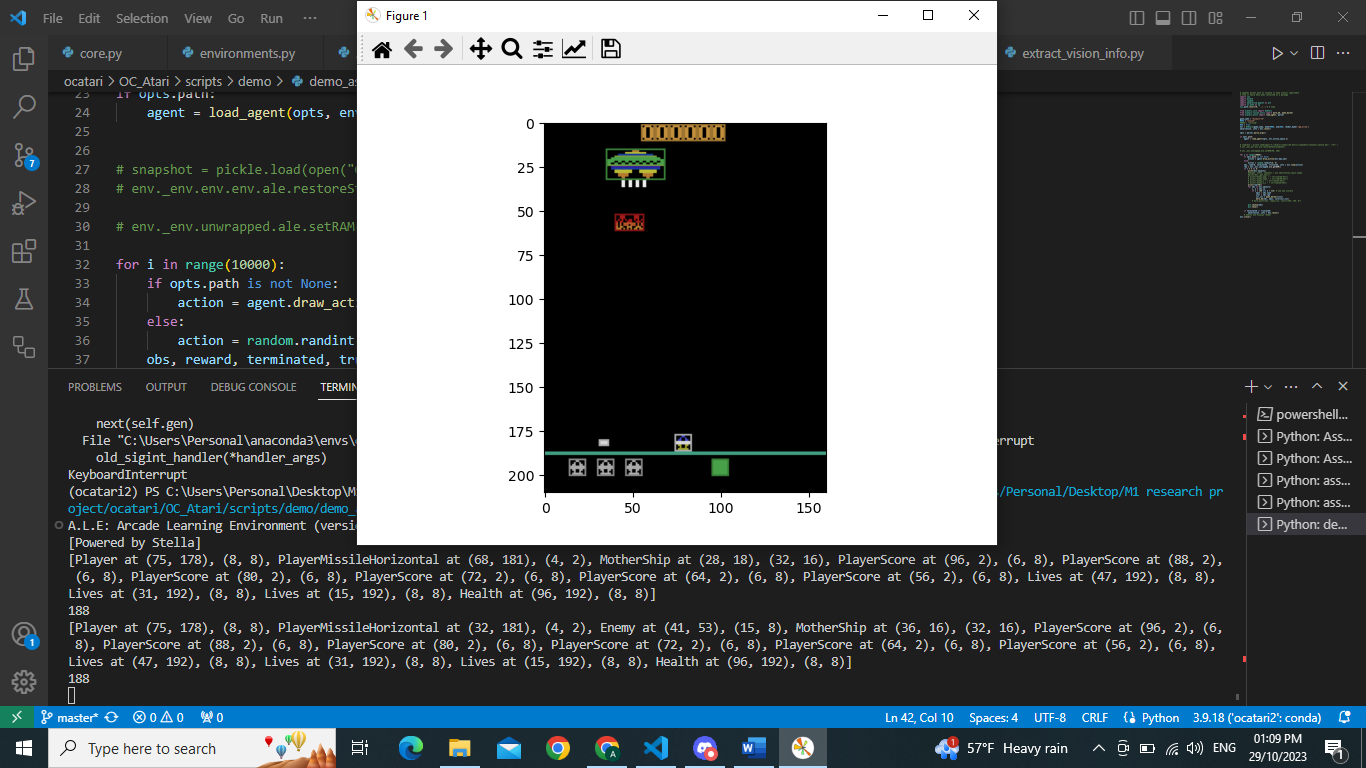
Also the player score has 6 objects 1 for each digit. Each life is also counted as an object.

******

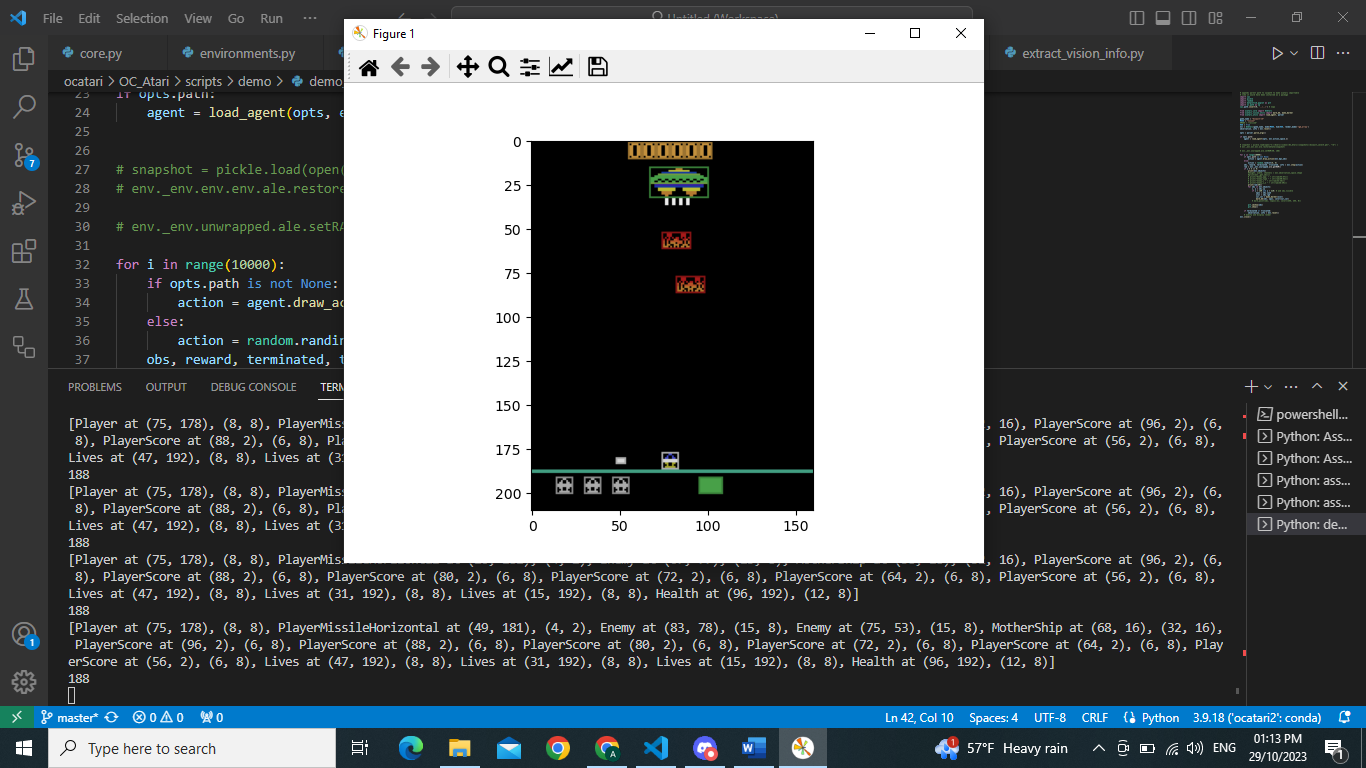
The second observation was made after w new enemy has appeared in the frame resulting in 14 objects on screen:

* Player at (75, 178), (8, 8),
* PlayerMissileHorizontal at (32, 181), (4, 2),
* **Enemy** at (41, 53), (15, 8),
* MotherShip at (36, 16), (32, 16),
* PlayerScore at (96, 2), (6, 8), PlayerScore at (88, 2), (6, 8), PlayerScore at (80, 2), (6, 8), PlayerScore at (72, 2), (6, 8), PlayerScore at (64, 2), (6, 8), PlayerScore at (56, 2), (6, 8),
* Lives at (47, 192), (8, 8), Lives at (31, 192), (8, 8), Lives at (15, 192), (8, 8),
* Health at (96, 192), (8, 8)]

Notice how we see that the array size increases with each new object that appears on the screen(here the new enemy)



Another enemy appears:



***2)USING “RAW”:***

As we can see here the initial observation was made when we had

Number of objects: 8

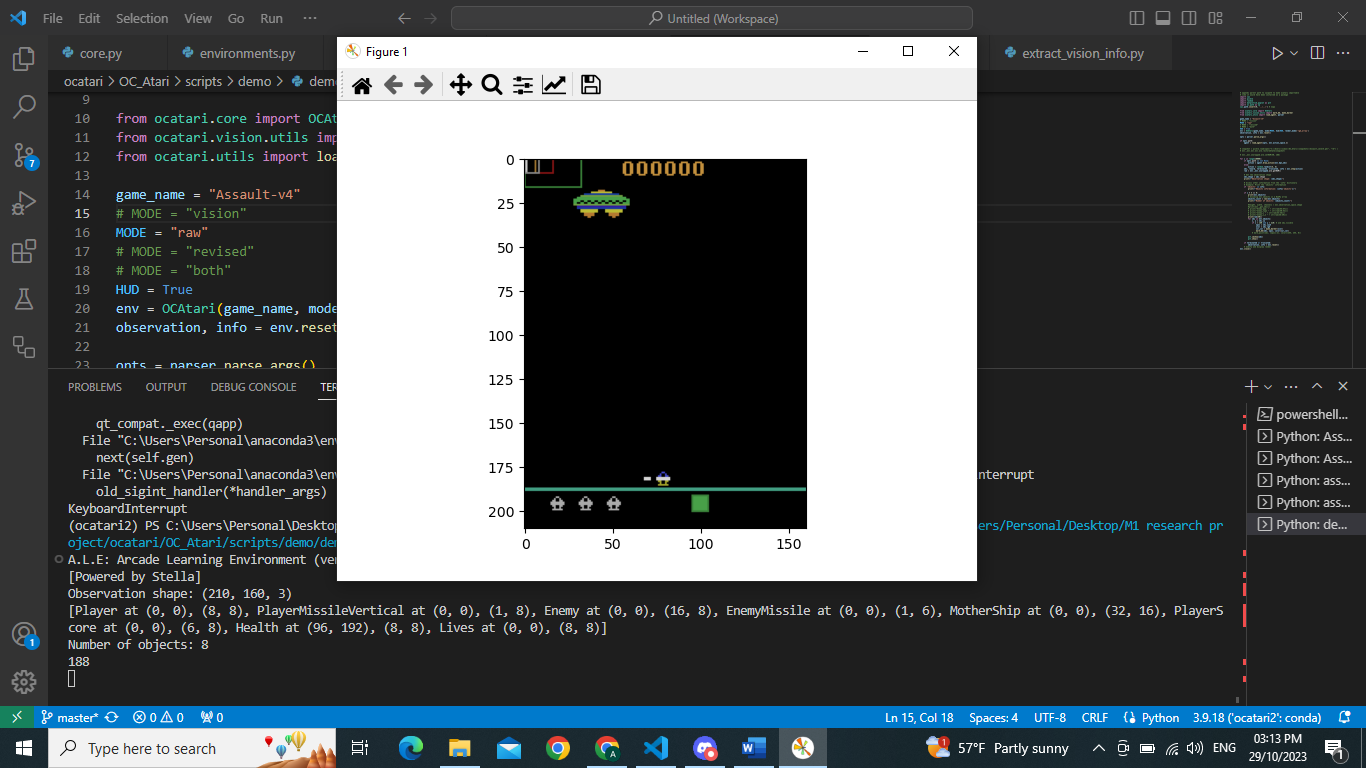
Observation shape: (210, 160, 3)

* [Player at (0, 0), (8, 8),
* PlayerMissileVertical at (0, 0), (1, 8),
* Enemy at (0, 0), (16, 8),
* EnemyMissile at (0, 0), (1, 6),
* MotherShip at (0, 0), (32, 16),
* PlayerScore at (0, 0), (6, 8),
* Health at (96, 192), (8, 8),
* Lives at (0, 0), (8, 8)]

First 2 digits being the coordinates, the second two being their dimensions.

We have “Enemy at” and “EnemyMissile at” even though we don’t have them on screen yet.

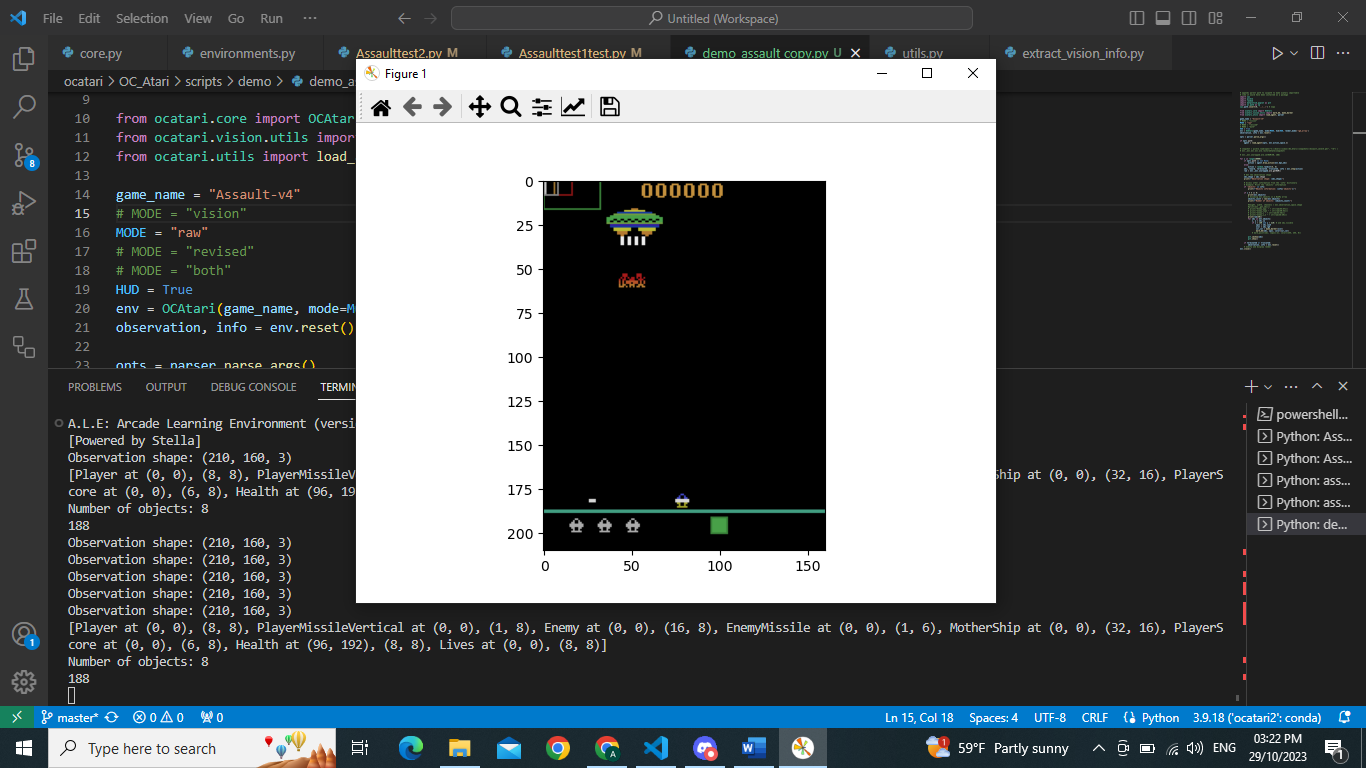
Notice how we cannot keep track of the position of each object.



Observation shape: (210, 160, 3)

Number of objects: 8

* [Player at (0, 0), (8, 8),
* PlayerMissileVertical at (0, 0), (1, 8),
* Enemy at (0, 0), (16, 8),
* EnemyMissile at (0, 0), (1, 6),
* MotherShip at (0, 0), (32, 16),
* PlayerScore at (0, 0), (6, 8),
* Health at (96, 192), (8, 8),
* Lives at (0, 0), (8, 8)]



***3)USING “REVISED”:***

As we can see here the initial observation was made when we had

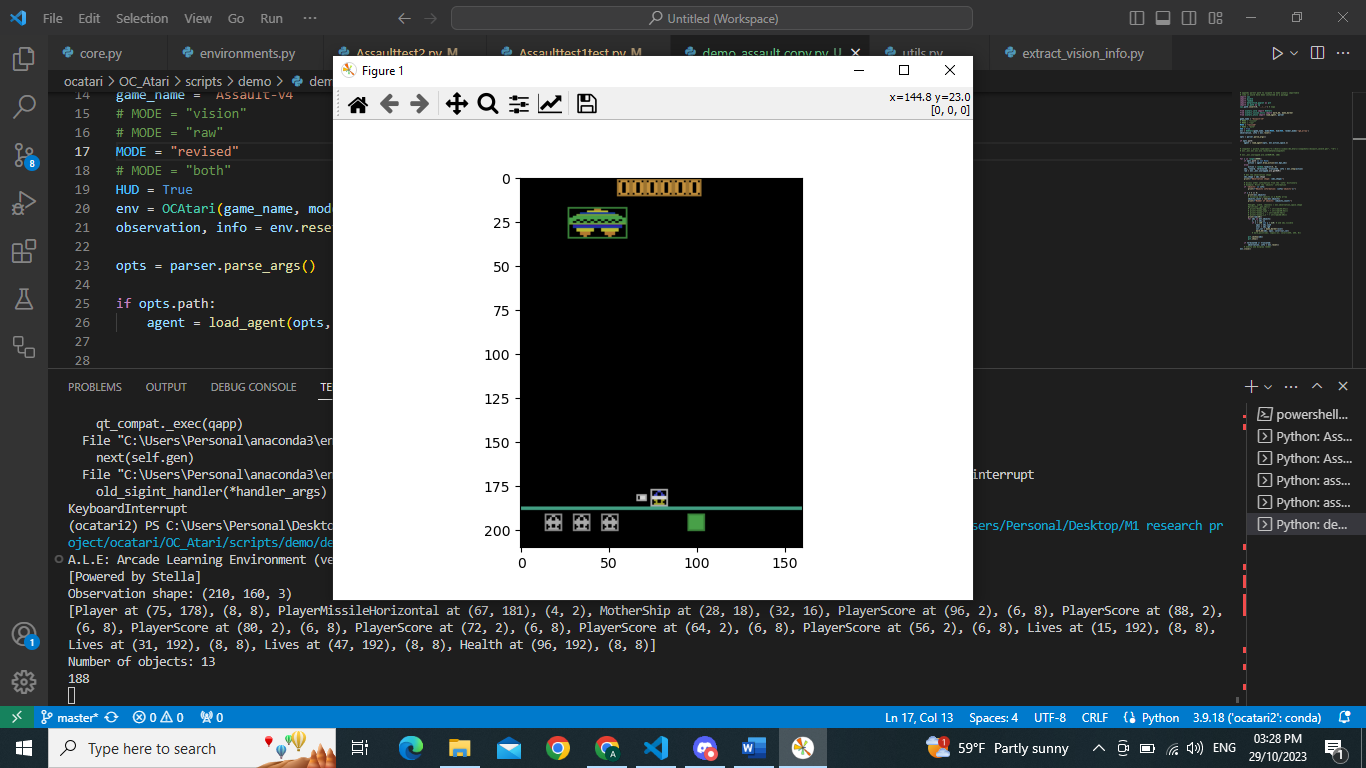
Number of objects: 13

Observation shape: (210, 160, 3)

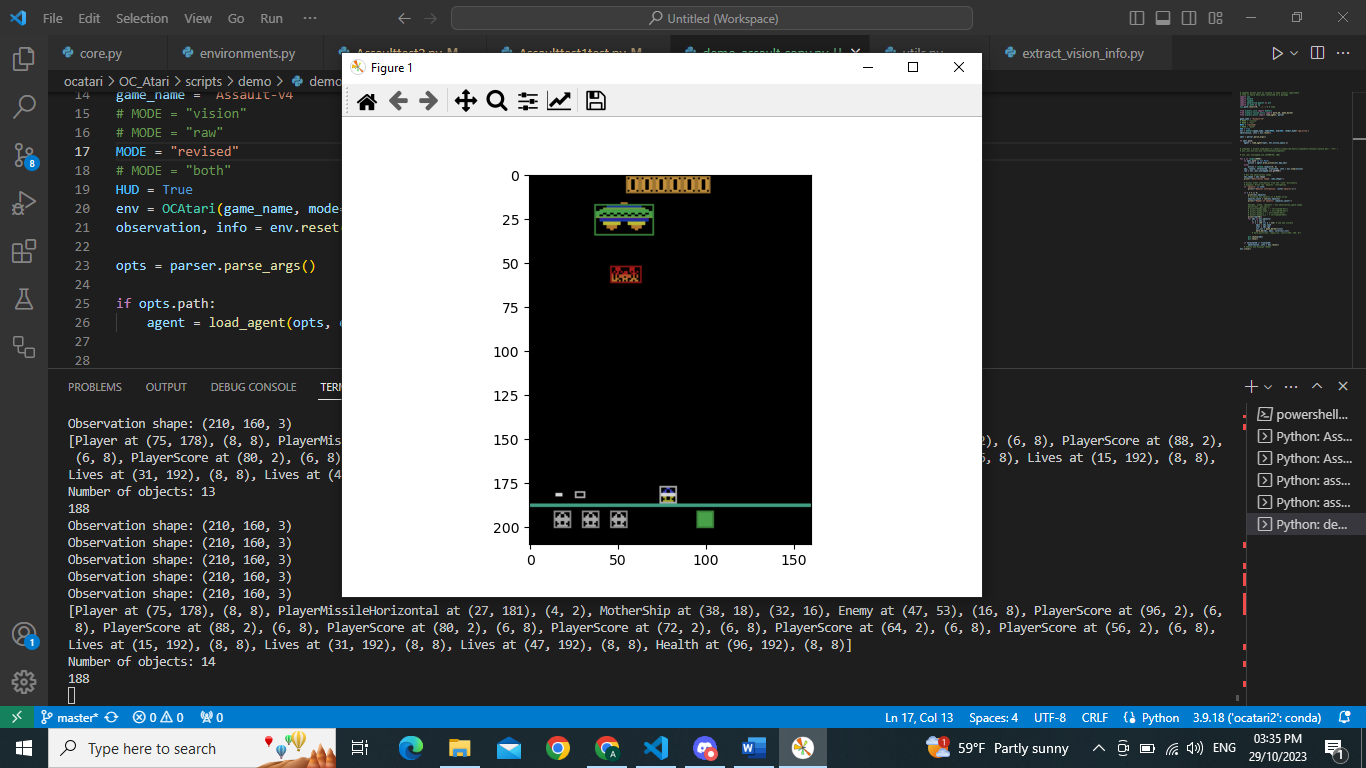
* [Player at (75, 178), (8, 8),
* PlayerMissileHorizontal at (67, 181), (4, 2),
* MotherShip at (28, 18), (32, 16),
* PlayerScore at (96, 2), (6, 8), PlayerScore at (88, 2), (6, 8), PlayerScore at (80, 2), (6, 8), PlayerScore at (72, 2), (6, 8), PlayerScore at (64, 2), (6, 8), PlayerScore at (56, 2), (6, 8),
* Lives at (15, 192), (8, 8), Lives at (31, 192), (8, 8), Lives at (47, 192), (8, 8),
* Health at (96, 192), (8, 8)]

First 2 digits being the coordinates, the second two being their dimensions.

Very similar to using “vision”. player score has 6 objects 1 for each digit. Each life is also counted as an object, but for some reason live are counted backwards.



Similar behavior however; once a new object appears, it adds it.



***4)USING “BOTH”:***

As we can see here the initial observation was made when we had

Number of objects: 13

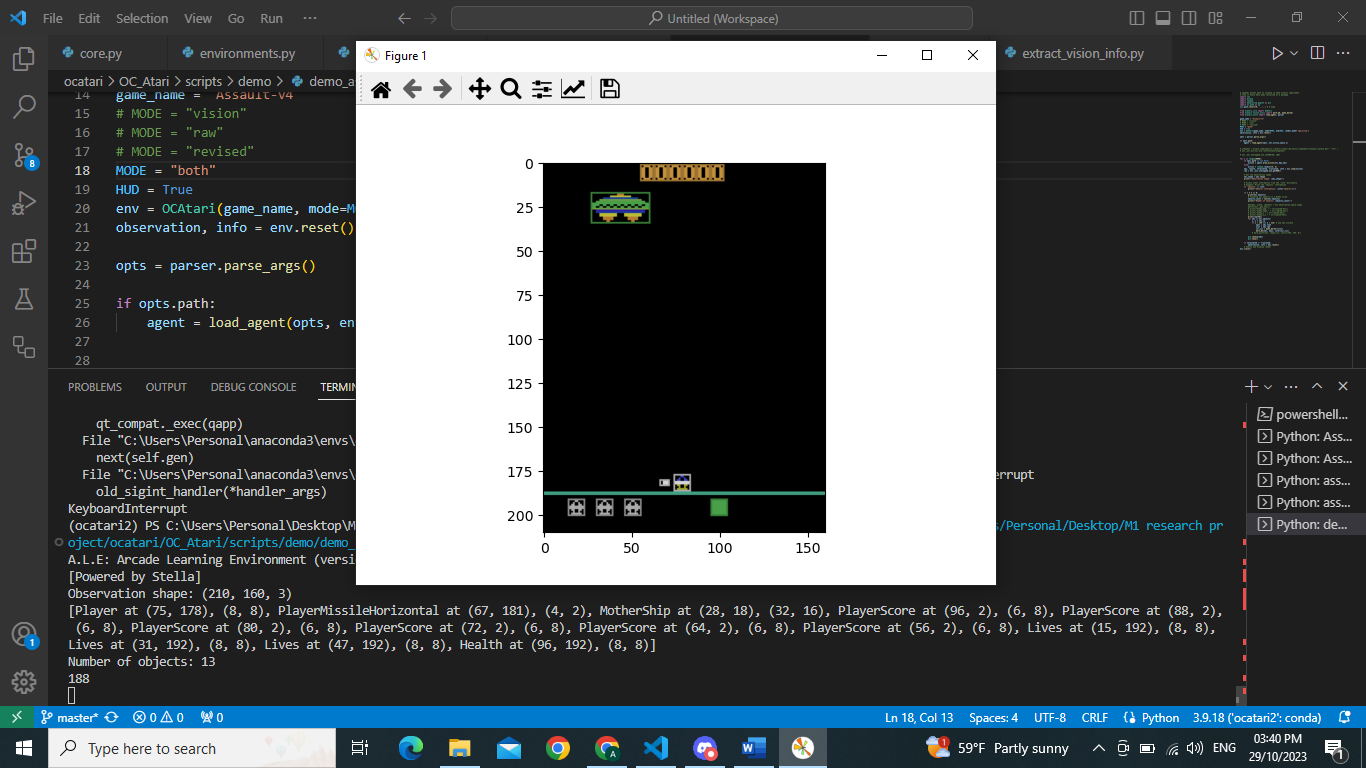
Observation shape: (210, 160, 3)

* [Player at (75, 178), (8, 8), PlayerMissileHorizontal at (67, 181), (4, 2), MotherShip at (28, 18), (32, 16), PlayerScore at (96, 2), (6, 8), PlayerScore at (88, 2), (6, 8), PlayerScore at (80, 2), (6, 8), PlayerScore at (72, 2), (6, 8), PlayerScore at (64, 2), (6, 8), PlayerScore at (56, 2), (6, 8), Lives at (15, 192), (8, 8),

Lives at (31, 192), (8, 8), Lives at (47, 192), (8, 8), Health at (96, 192), (8, 8)]

Number of objects: 13First 2 digits being the coordinates, the second two being their dimensions.

Very similar to using “vision”. player score has 6 objects 1 for each digit. Each life is also counted as an object, but for some reason live are counted backwards.

  
Number of objects: 14

* [Player at (75, 178), (8, 8),
* PlayerMissileHorizontal at (27, 181), (4, 2),
* MotherShip at (36, 18), (32, 16),
* Enemy at (41, 53), (16, 8),
* PlayerScore at (96, 2), (6, 8), PlayerScore at (88, 2), (6, 8), PlayerScore at (80, 2), (6, 8), PlayerScore at (72, 2), (6, 8), PlayerScore at (64, 2), (6, 8), PlayerScore at (56, 2), (6, 8),
* Lives at (15, 192), (8, 8), Lives at (31, 192), (8, 8), Lives at (47, 192), (8, 8),
* Health at (96, 192), (8, 8)]

