



GDAI

Playing Geometry Dash with Deep Q Reinforcement Learning



Challenges

01

Reading in the screen

The model needed something to interpret and other than reading in the memory, reading in the display is the only way.

02

Detecting death

Without an environment, I had to find a way to detect when the AI dies to know when to move on to the next episode.

03

Input (Jumping)

There is only one input in GD which is pressing the mouse button to jump. You can also hold to continuously jump.

04

Reward Metric

The main goal of the game is to go as far as possible in one level.

Reading in the Screen

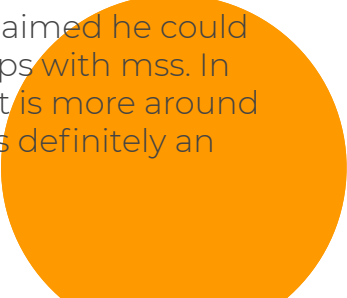


PIL: ImageGrab

ImageGrab is the first display reading library I tried. It worked after a little tuning, but when I tried to run the AI with it, I realized it took a long time to read in the display. This resulted in an fps of about 3 because GD runs at the same speed.

mss

To solve the low fps problem, I looked for a faster display grabbing library. One user on stack overflow claimed he could get a stable 60 fps with mss. In my experience, it is more around ~15 fps but it was definitely an upgrade from 3.



To make the image easier to read in for the CNN, I cropped it to only include useful information, changed it to grayscale, and did edge detection.



(I'm still not sure if edge detection actually helps)



```
def get_screen():  
    with mss.mss() as sct:  
        screen = np.array(sct.grab(monitor))  
        # Simplify image  
        screen = cv2.cvtColor(screen, cv2.COLOR_BGR2GRAY)  
        screen = cv2.Canny(screen, threshold1 = 200, threshold2=300)  
        return screen
```

Detecting Death

I detected images by continuously comparing the previous frame to the current frame and seeing if their similarity was $< 99.9\%$ using `compare_ssim` from `sklearn.image`. If the similarity was below 99.9% the the bot is still alive. I also used a frame queue to make sure it was checking older frames.



Dead

```
from pynput.mouse import Button, Controller
import time
from Helpers import get_screen, isalive

mouse = Controller()

def press_LB():
    mouse.position = (180, 350)
    mouse.press(Button.left)

def release_LB():
    mouse.position = (180, 350)
    mouse.release(Button.left)

def restart():
    mouse.position = (180, 425)
    death_screen = get_screen()
    screen = death_screen
    while not isalive(screen, death_screen):
        mouse.click(Button.left, 1)
        screen = get_screen()
    time.sleep(0.25)
```



Input (Jumping)

Used the pynput library to click
with a mouse

Reward Metric

There were three possible reward metrics I tried

- Number of rounds
- Penalize whenever you jump
- Penalize whenever you change input

I wanted to penalize jumping to make the bot learn to play like an actual player that only jumps when necessary. For changing inputs, geometry dash has a low click challenge for every level. (I ended up just using the time alive)



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 201, 231, 32)	2624
activation (Activation)	(None, 201, 231, 32)	0
max_pooling2d (MaxPooling2D)	(None, 100, 115, 32)	0
conv2d_1 (Conv2D)	(None, 48, 56, 64)	51264
activation_1 (Activation)	(None, 48, 56, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 24, 28, 64)	0
conv2d_2 (Conv2D)	(None, 11, 13, 128)	73856
activation_2 (Activation)	(None, 11, 13, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 5, 6, 128)	0
conv2d_3 (Conv2D)	(None, 3, 4, 256)	295168
activation_3 (Activation)	(None, 3, 4, 256)	0
conv2d_4 (Conv2D)	(None, 1, 2, 512)	1180160
activation_4 (Activation)	(None, 1, 2, 512)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
activation_5 (Activation)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1026
Total params: 2,128,898		
Trainable params: 2,128,898		
Non-trainable params: 0		

CNN

(With mostly arbitrary parameters)



```
act = 'elu'
opt = RMSprop()

model = Sequential()
model.add(Conv2D(32,(9,9),strides=2, input_shape=(410,470,1)))
model.add(Activation(act))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64,(5,5),strides=2))
model.add(Activation(act))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128,(3,3),strides=2))
model.add(Activation(act))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256,(3,3)))
model.add(Activation(act))

model.add(Conv2D(512,(3,3)))
model.add(Activation(act))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation(act))

model.add(Dense(self.action_size, activation='linear'))
model.compile(loss='mse', optimizer=opt)
```




Results

My original goal was to reach the first ship part in Stereo Madness. My best score was 12% through which was about a third of the way to the ship part.



Possible **Future** Features to Implement

Object Recognition

Instead of giving the neural network the display I could give it the sizes and position of objects relative to itself.

Better Jumping

Only possible with object recognition, but I could make sure the neural network does not make any decisions that are not used (when player is in the air)

Randomized Levels

To prevent overfitting on just Stereo Madness