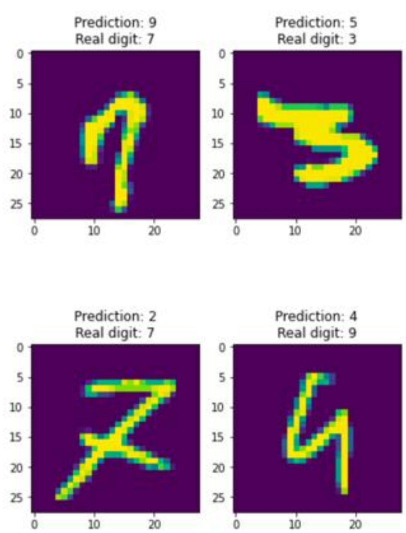
## Computer Vision

CNN, Adversarial Attack, Deep Learning on 3D Data, Camera Model and Rigid Transformations

Works I have done in this project:

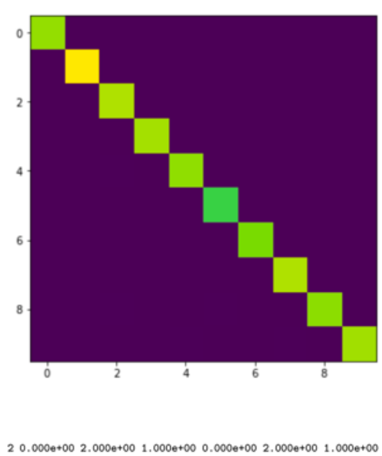
- Implemented convolutional neural network on handwritten digit database (CNN)
  - Designed layers in the model to achieve 99% test accuracy
  - Plotted confusion matrix to measure the performance of the network
  - Visualized weights and kernels of the first layer of the CNN
- Adversarial Attack
  - Generated adversarial noise using the fast sign gradient method
  - Added noise to the input image (from handwritten digit database)
  - Plotted output images added with different level of noise by adversarial attack
- Deep Learning on 3D Data (dataset with classes chair, car, lamp, airplane and table)
  - Built pointnet architecture to classify data to classes with at least 90% accuracy
  - Plotted critical points / salient points of data
- Camera Model and Rigid Transformations
  - Viewed points from a camera model with different focal length and optical axis



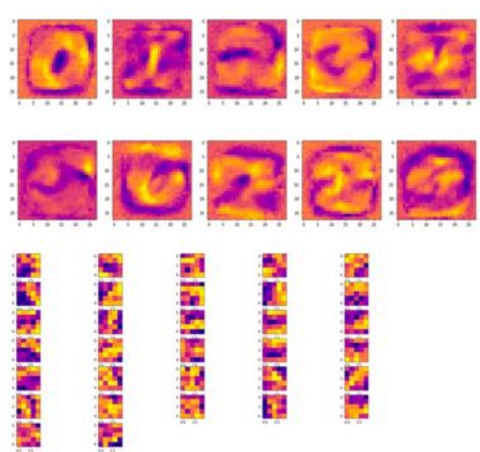

*Plot digits that the network got wrong*

```
class CNN(nn.Module):
    def __init__(self, input_size, num_classes):
        """
        init convolution and activation layers
        Args:
            input_size: (1,28,28)
            num_classes: 10
        """
        super(CNN, self).__init__()

        ### YOUR CODE HERE
        self.layer1 = nn.Conv2d(1, 32, kernel_size=5, padding=1)
        self.layer2 = nn.Conv2d(32, 64, kernel_size=5)
        self.flat = nn.Flatten()
        self.fc1 = nn.Linear(1024, 10)
        ### END OF CODE


    def forward(self, x):
        """
        forward function describes how input tensor is transformed to output␣
↪tensor
        Args:
            x: (Nx1x28x28) tensor

        """
        ### YOUR CODE HERE
        # x = torch.flatten(x, 1)
        # print(x.shape)
        x = F.max_pool2d(F.relu(self.layer1(x)), 2)
        x = F.max_pool2d(F.relu(self.layer2(x)), 2)
        x = self.flat(x)
        # print(x.shape)
        x = self.fc1(x)
        ### END OF CODE
        return x
```

*CNN architecture*




*Confusion Matrix*



*Visualize weights and kernels*

```
accuracies = []
examples = []
epsilons = [0, .05, .1, .15, .2, .25, .3]
# put model in eval mode
CNNTrainer.model.eval()

# update the helper class with a new batch size = 1
# for the dataloader. It doesnt change the model
# weights or any other parameter

opts = {
    'lr': 5e-4,
    'epochs': 5,
    'batch_size': 1 #this is the only change
}

CNNTrainer = TrainHelper(model = CNNTrainer.model,
                         train_set = train_dataset,
                         test_set = test_dataset,opts = opts)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Run test for each epsilon
for eps in epsilons:
    acc, ex = eval_adversarial(CNNTrainer, device, eps)
    accuracies.append(acc)
    examples.append(ex)
```

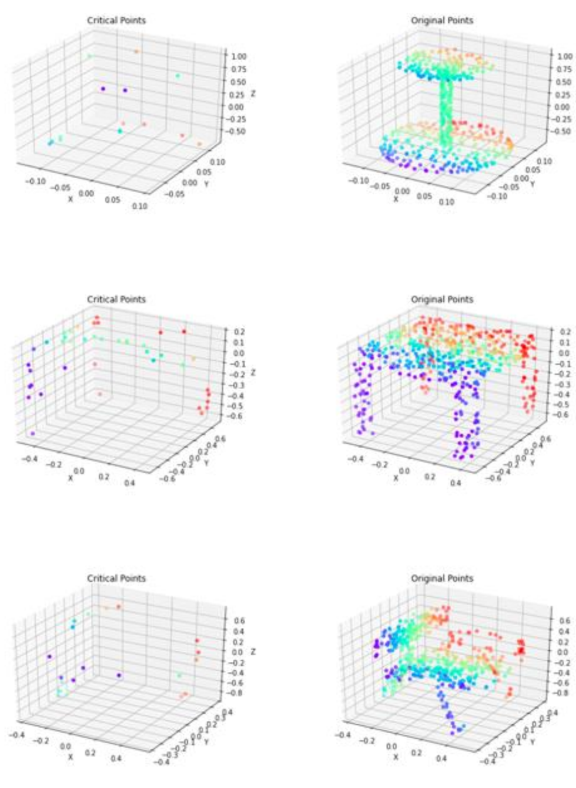
```
Epsilon: 0      Test Accuracy = 9919 / 10000 = 0.9919
Epsilon: 0.05   Test Accuracy = 9537 / 10000 = 0.9537
Epsilon: 0.1    Test Accuracy = 8328 / 10000 = 0.8328
Epsilon: 0.15   Test Accuracy = 5710 / 10000 = 0.571
Epsilon: 0.2    Test Accuracy = 3335 / 10000 = 0.3335
Epsilon: 0.25   Test Accuracy = 1896 / 10000 = 0.1896
Epsilon: 0.3    Test Accuracy = 1093 / 10000 = 0.1093
```

*Test Accuracy with different level of adversarial attack*


*Output images with noise*

```
crit_point_dict = {}
label_indices = [0,1,2,3,4]
criterion = torch.nn.CrossEntropyLoss() # this allows you to not need the TrainHelper Class

for i, (data, labels) in enumerate(data_loader):
    # data.shape = [1, 500, 3]
    # labels.shape = [1]
    if labels.numpy() in label_indices:   # print one sample from each class
        # create index in the dictionary and remove the class id from the list of indices
        # we are looking for
        class_label = labels.numpy()[0]   # get class of current point cloud
        label_indices.remove(class_label) # we don't print more samples from that class
        crit_point_dict[class_label] = {}
        crit_point_dict[class_label]['indices'] = []
        crit_point_dict[class_label]['data'] = []

        # run sample through the network


        ### START CODE HERE ###
        data.requires_grad = True
        model.zero_grad()
        output = PtNet.forward(data)
        # ouput.shape = [1, 5]
        pred = output.max(1, keepdim=True)[1]
        loss = criterion(output, labels)
        loss.backward()
        print(loss)
        data_grad = data.grad.data
        # data_grad.shape = [1, 500, 3]
        # data_grad.squeeze(0).shape = [500, 3]
        data_squ = data_grad.squeeze(0)
        for i in range(data_squ.shape[0]):
          if data_squ[i][0].item() > 0 or data_squ[i][1].item() > 0 or data_squ[i][2].item() > 0:
            crit_point_dict[class_label]['indices'].append(i)

        crit_point_dict[class_label]['data'] = data.squeeze(0).detach().numpy()
```

*Partial code of finding critical points in 3D data*



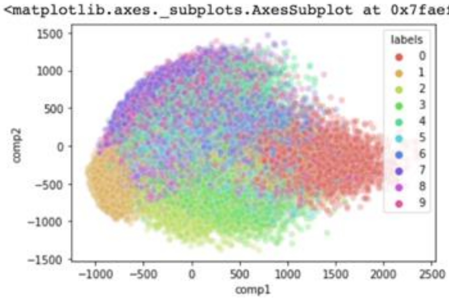

*Critical Points vs Original Points*

## [PCA, NN, CNN (Python with numpy, pytorch, matplotlib, pandas, sklearn)]

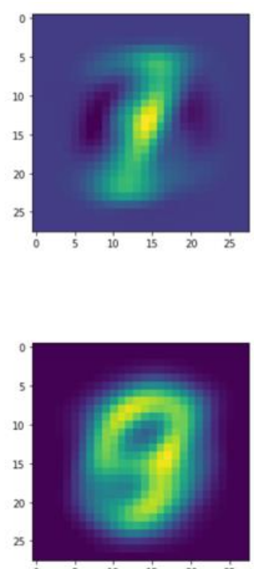
Works I have done in this project:

- Preprocess dataset with Principal Component Analysis (PCA)
    - Perform PCA using 2 methods: singular value decomposition and eigen decomposition
    - Project data to different dimensions to see how well the digits can be represented after PCA
- Neural Network for regression
    - Build a basic MultiLayer Perceptron (MLP) for fitting a line to random data
    - Decide on the right optimizer, choose an appropriate loss function for my learning task, and train the model over the given data
- Corner Detection using Convolution Neural Network (CNN)
    - The CNN will output the coordinates of the corner in the given image with a corner in it
    - Build convolution and maxpool layers, and pick non-linear activation function for each layer in the model

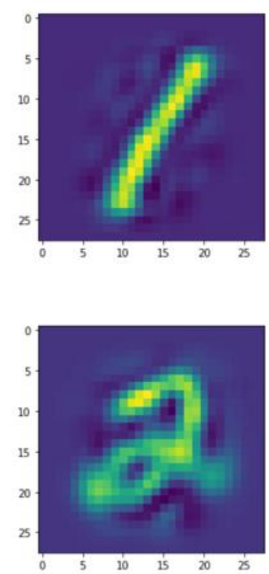
```
[8] ### YOUR CODE HERE
    import seaborn as sns
    projected = np.matmul(data_centered, V.T[:,:2])
    df = pd.DataFrame(data = projected, columns=['comp1', 'comp2'])
    df['labels'] = pd.Series(labels)
    sns.scatterplot(
        x="comp1", y="comp2",
        hue="labels",
        palette=sns.color_palette("hls", 10),
        data=df,
        legend="full",
        alpha=0.3
    )
    ### END OF CODE

    <matplotlib.axes._subplots.AxesSubplot at 0x7faef9eecf98>
```




Data distribution in space spanned by 2 principal components

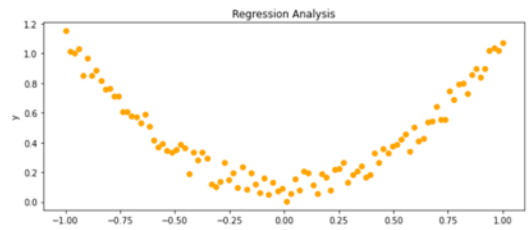


Data in 2 dimensions

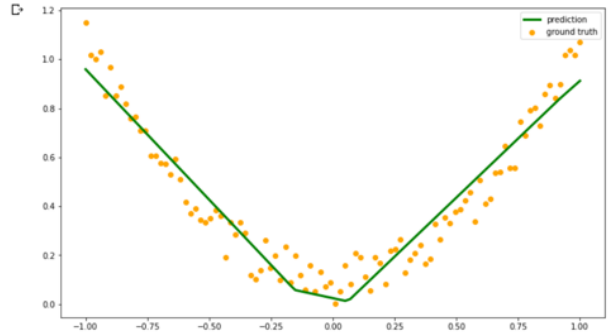


Data in 50 dimensions

```
torch.manual_seed(1)    # reproducible

x = torch.unsqueeze(torch.linspace(-1, 1, 100), dim=1)  # x data (tensor), shap
y = x.pow(2) + 0.2*torch.rand(x.size())                 # noisy y data (tensor)

# torch can only train on Variable, so convert them to Variable
x, y = Variable(x), Variable(y)

# view data
plt.figure(figsize=(10,4))
plt.scatter(x.data.numpy(), y.data.numpy(), color = "orange")
plt.title('Regression Analysis')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

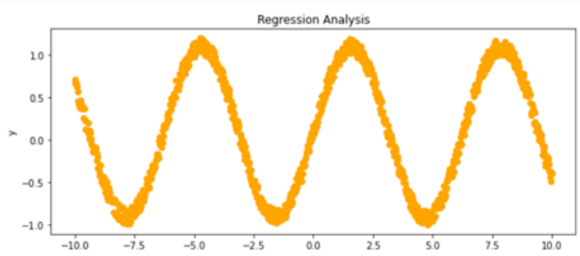



```
fig, ax = plt.subplots(figsize=(12,7))
ax.scatter(x.data.numpy(), y.data.numpy(), color = "orange", label='ground truth')
ax.plot(x.data.numpy(), prediction.data.numpy(), 'g-', lw=3, label='prediction')
plt.legend()
plt.show()
```

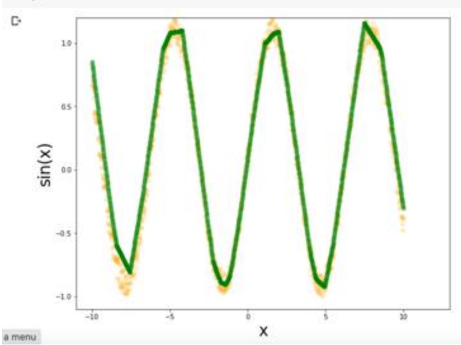



*Neural Network for regression*

```
[20]
x = torch.unsqueeze(torch.linspace(-10, 10, 1000), dim=1)  # x data (tensor),
y = torch.sin(x) + 0.2*torch.rand(x.size())                 # noisy y data (t

x, y = Variable(x), Variable(y)

plt.figure(figsize=(10,4))
plt.scatter(x.data.numpy(), y.data.numpy(), color = "orange")
plt.title('Regression Analysis')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



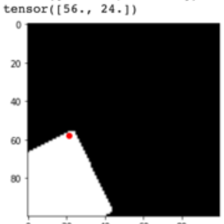

```
fig, ax = plt.subplots(figsize=(10,8))
# ax.set_title('Regression Analysis - model 3, Batches', fontsize=35)
ax.set_xlabel('x', fontsize=24)
ax.set_ylabel('sin(x)', fontsize=24)
ax.set_xlim(-11.0, 13.0)
ax.set_ylim(-1.1, 1.2)
ax.scatter(x.data.numpy(), y.data.numpy(), color = "orange", alpha=0.2)
prediction = net(x)     # input x and predict based on x
ax.scatter(x.data.numpy(), prediction.data.numpy(), color='green', alpha=0.5)
plt.show()
```



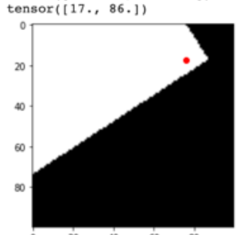

*Implement MLP to fit the sine function*

```
with torch.no_grad():
  for i,(x,y) in enumerate(testloader):
    x = torch.unsqueeze(x, 1)
    coords = net(x)

    for j in range(10):
      fig, ax = plt.subplots()
      ax.imshow(x[j][0], cmap=plt.cm.gray)
      pts = y[j]
      pts = coords[j] * 100
      print(pts)
      print(y[j])
      ax.plot(pts[1], pts[0], color='red', marker='o',
              linestyle='None', markersize=6)
      plt.show()

    break
```

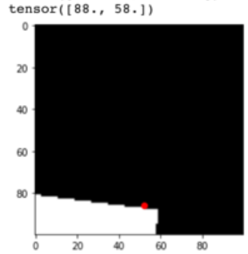
```
tensor([57.6018, 21.0295])
tensor([56., 24.])
```



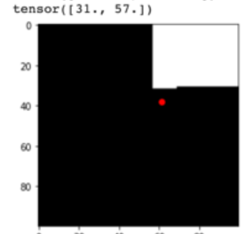
```
tensor([17.1382, 75.7792])
tensor([17., 86.])
```



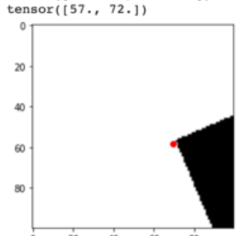
```
tensor([86.0042, 52.0330])
tensor([88., 58.])
```



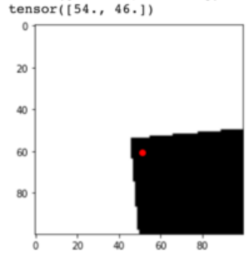
```
tensor([37.9298, 61.1929])
tensor([31., 57.])
```



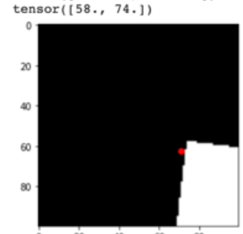
```
tensor([58.1141, 69.4779])
tensor([57., 72.])
```



```
tensor([60.5148, 50.9567])
tensor([54., 46.])
```



```
tensor([62.5693, 71.0829])
tensor([58., 74.])
```



*Corner Detection*

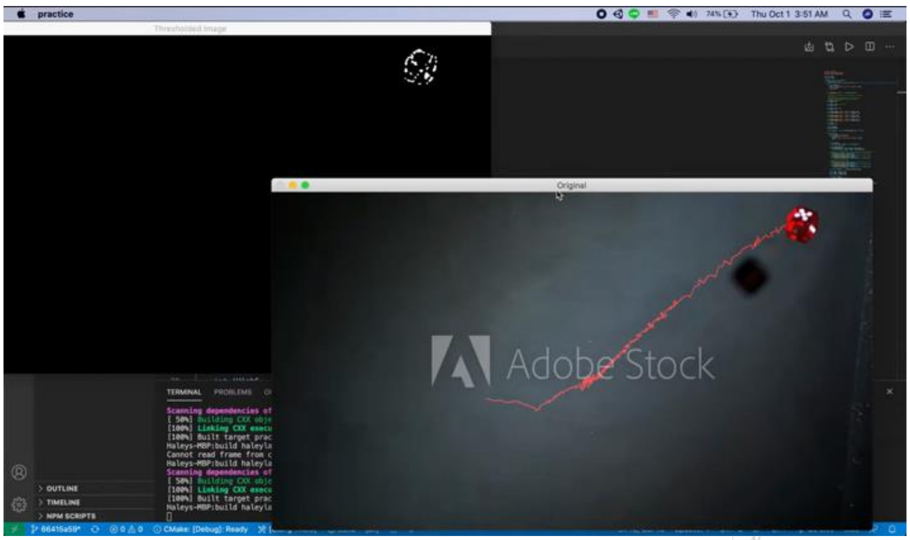
## Object Tracking (OpenCV with C++)

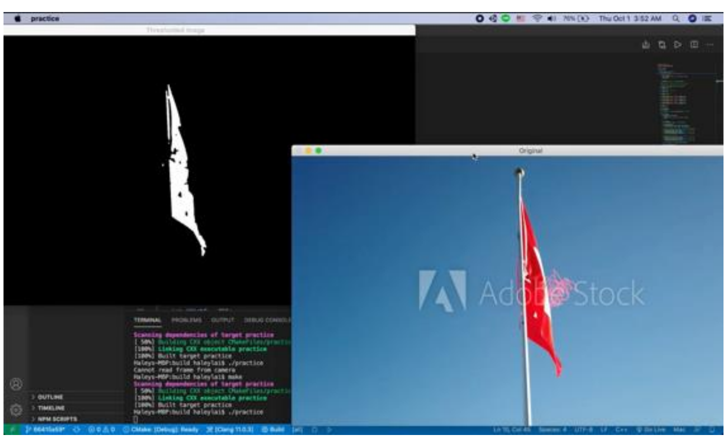
- A simple object tracking using color and shape

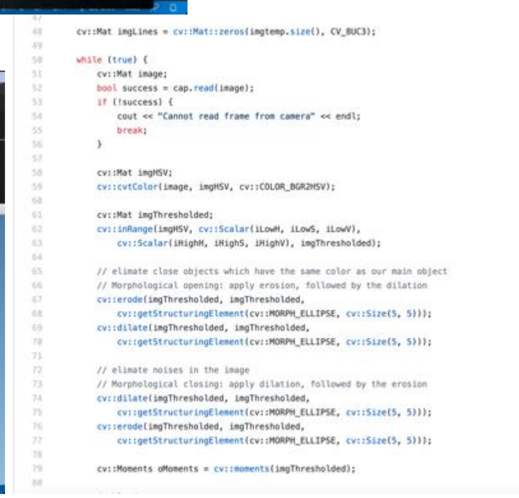
In the following example, the program is targeting red object specifically

What does the program do:

- Thresholded Image (left window in the figures):
  - By applying multiple image filters on each frame of the video, the program is able to segment out the main object in the video.
  - All unrelated backgrounds or objects will be filtered out during the image processing step (image filters)
- Original Image (right window in the figures):
  - Find the exact position of the target object in each frame
  - Draw a line along the trajectory of the object



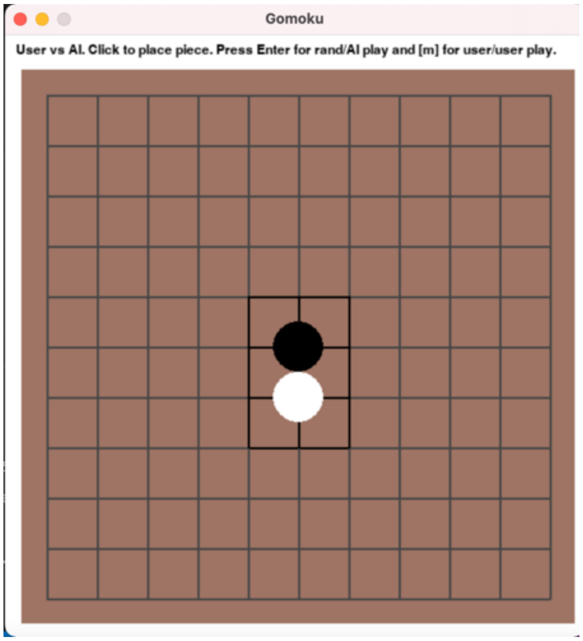



```cpp
cv::Mat imgLines = cv::Mat::zeros(imgtemp.size(), CV_8UC3);

while (true) {
    cv::Mat image;
    bool success = cap.read(image);
    if (!success) {
        cout << "Cannot read frame from camera" << endl;
        break;
    }

    cv::Mat imgHSV;
    cv::cvtColor(image, imgHSV, cv::COLOR_BGR2HSV);

    cv::Mat imgThresholded;
    cv::inRange(imgHSV, cv::Scalar(iLowH, iLowS, iLowV),
        cv::Scalar(iHighH, iHighS, iHighV), imgThresholded);

    // elimate close objects which have the same color as our main object
    // Morphological opening: apply erosion, followed by the dilation
    cv::erode(imgThresholded, imgThresholded,
        cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));
    cv::dilate(imgThresholded, imgThresholded,
        cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));

    // elimate noises in the image
    // Morphological closing: apply dilation, followed by the erosion
    cv::dilate(imgThresholded, imgThresholded,
        cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));
    cv::erode(imgThresholded, imgThresholded,
        cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));

    cv::Moments oMoments = cv::moments(imgThresholded);
```

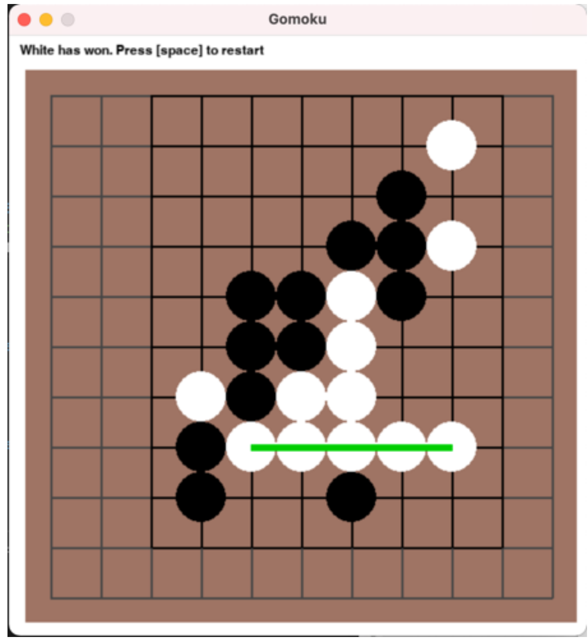
# AI: Search and Reasoning

## Gomoku AI (Python with pygame)

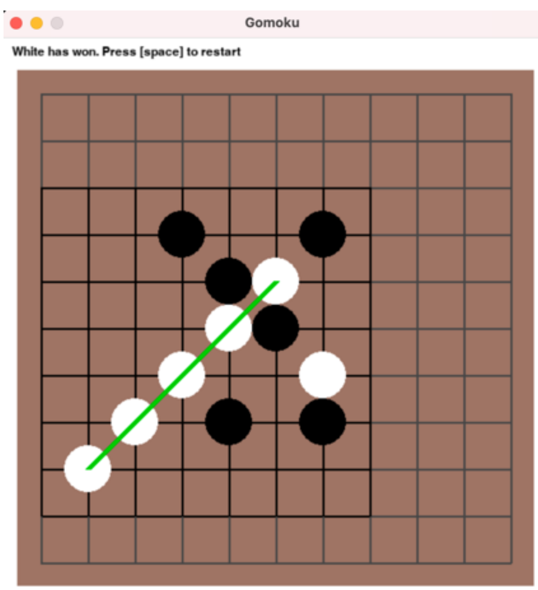
- Implement Monte Carlo Tree Search (MCTS) for playing Gomoku



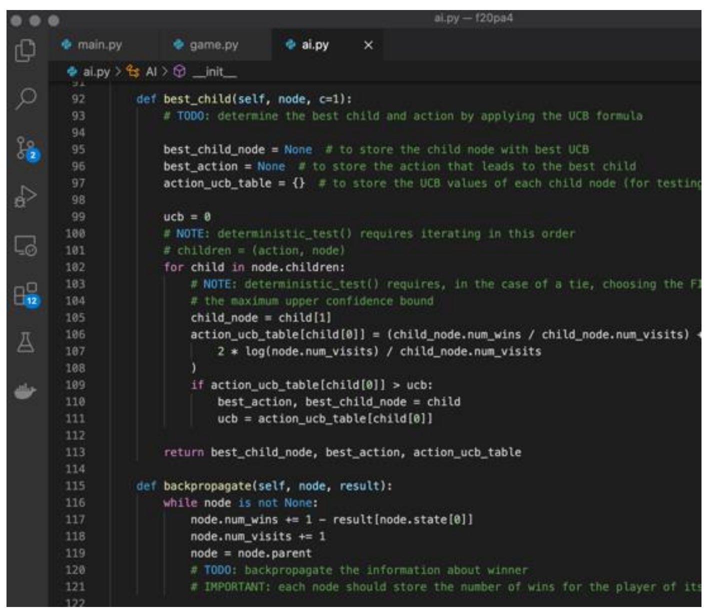

*Start Board*




*Player vs AI (AI won)*




*AI vs AI (White won)*




*Partial code of AI (picking best action out of samples derived from simulations)*

## Blackjack AI (Python with pygame)

- Implement Monte Carlo policy evaluation, Temporal Difference policy evaluation, and Q-Learning for Blackjack

**Description for each Evaluation:**

### Monte Carlo Policy Evaluation

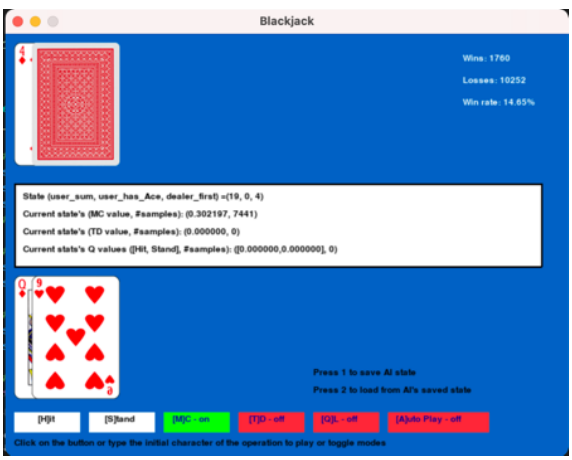
Evaluate the policy "Hit (ask for a new card) if sum of cards is below 14, and Stand (switch to dealer) otherwise" using the Monte Carlo method. Namely, learn the values for each state under the policy.

### Temporal-Difference Policy Evaluation

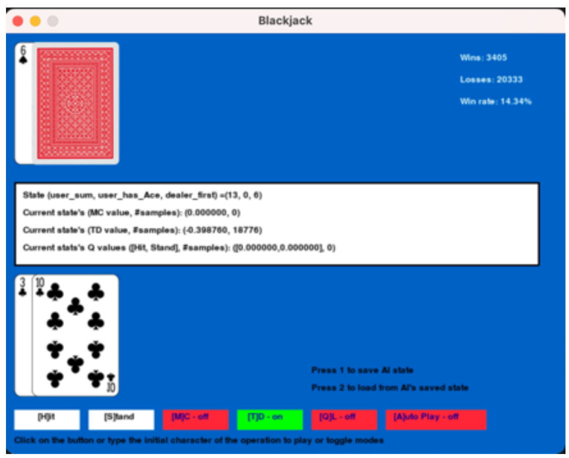
Evaluate the policy "Hit (ask for a new card) if sum of cards is below 14, and Stand (switch to dealer) otherwise" using the Temporal-Difference method.

### Q-Learning

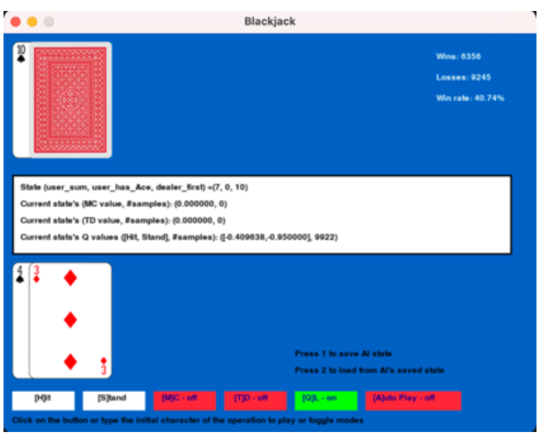
Implement the Q-learning algorithm. Use epsilon=0.4 in the epsilon-strategy in your final submission, but you are encouraged to check the behavior difference for various choices of epsilon. After learning, AutoPlay will follow the Q-learning values to make decisions.



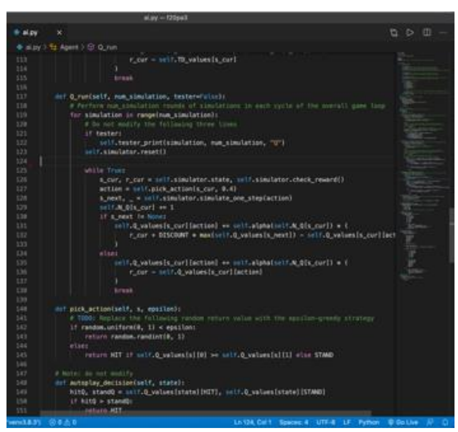

*Screenshot of game running under Monte Carlo Policy*




*Game running under Temporal-Difference Policy*



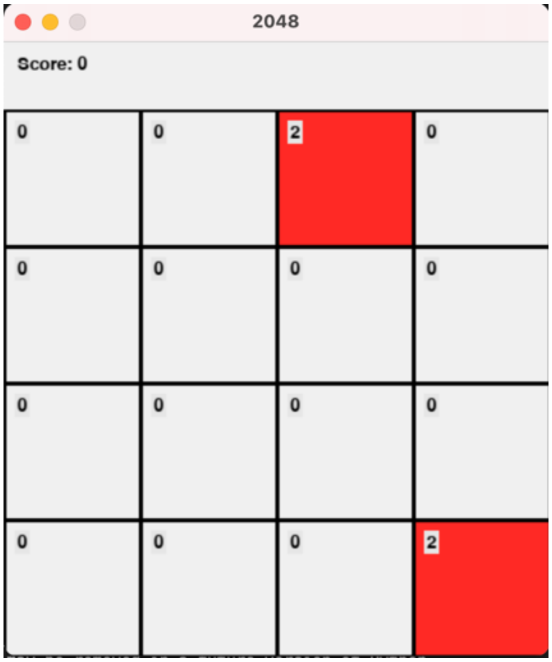

*Game running under Q-learning algorithm*



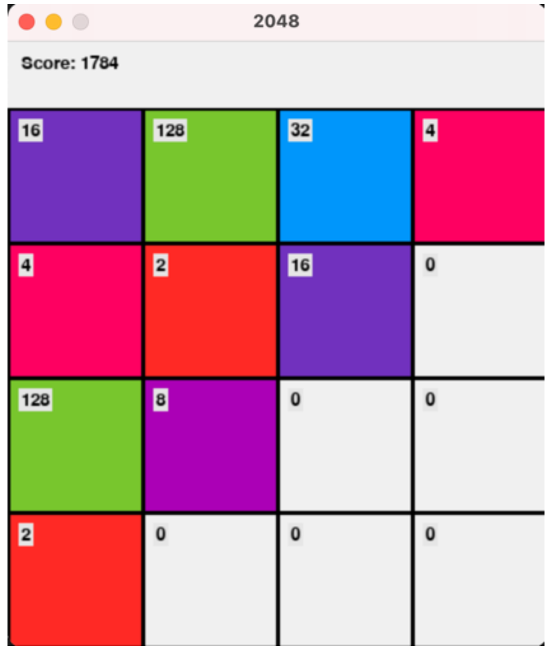
*Partial code of Q-learning algorithm*

## [2048 AI (Python with pygame)]

- Implement a game AI for the 2048 game based on expectimax search

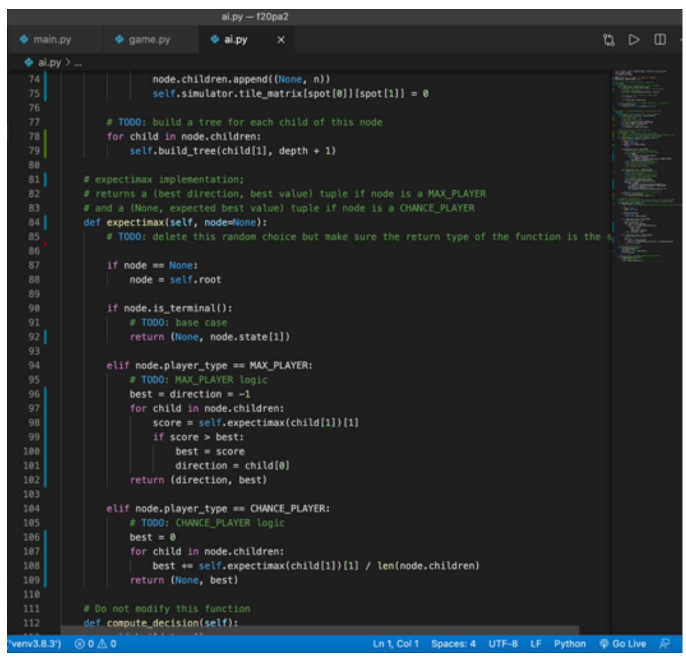



*Start board*




*Middle game*


*End game*




*Implementation code of expectimax algorithm*

## Grid World (Python with pygame)

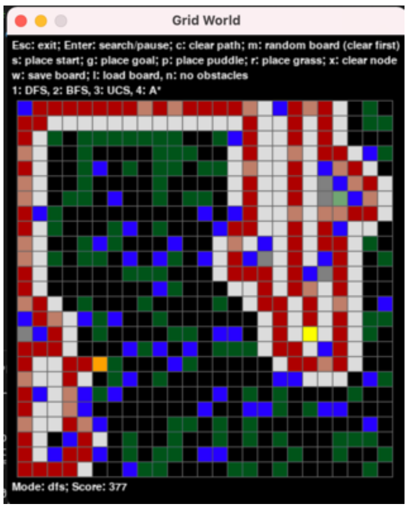
- Find paths from the start (yellow node) to the goal (orange node)

The program implements the following search strategies:

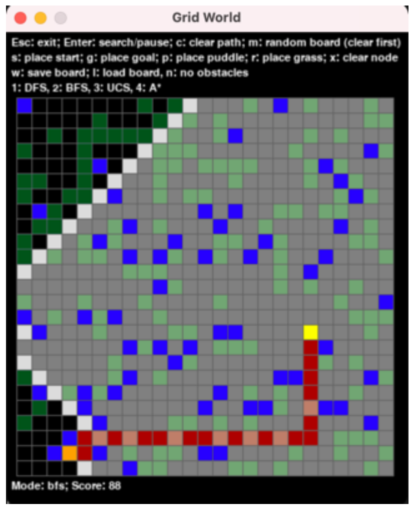
- DFS
- BFS
- Uniform Cost Search (Dijkstra)
- A* Search using Manhattan Distance as the heuristic

**Representation of colored grid in the figure:**

Puddle (Blue grid): player cannot go pass it
Grass (Green grid): has 10 costs
Start (Yellow grid): start point
Goal (Orange grid): destination




*DFS mode*




*BFS mode*


*Uniform Cost Search mode*


*A* mode*

*Partial code of A* implementation*

# 2020 Summer Internship – Game Development

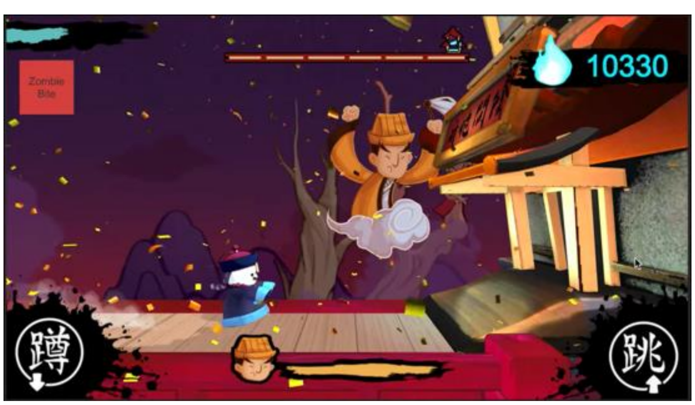
## [Zombie Runner Game (C# with Unity)]

- A classic 2D runner game made by Unity

How to play:

- Hitting slide and jump buttons to dodge obstacles
- Filling Zombie Bite by absorbing spirits in the game, use the skill to defend attacks from Daoshi
- Survive to the end





## Zombie Downstairs Game (C# with Unity)

- A classic 2D platform game made by Unity

How to play:

- Moving zombie to left and right to fall from platform to platform to not be hurt by the lightning trap from above
- Collecting enough spirits from human by attacking them to pass the game


**Game Interface**:
**Movement**: moving Zombie to left or right by hitting the left / right side of the screen

**Health bar** (top left): will reduce when hitting an obstacle. Game over when it reaches zero

**Human spirits** (top right): number of spirits the zombie has collected

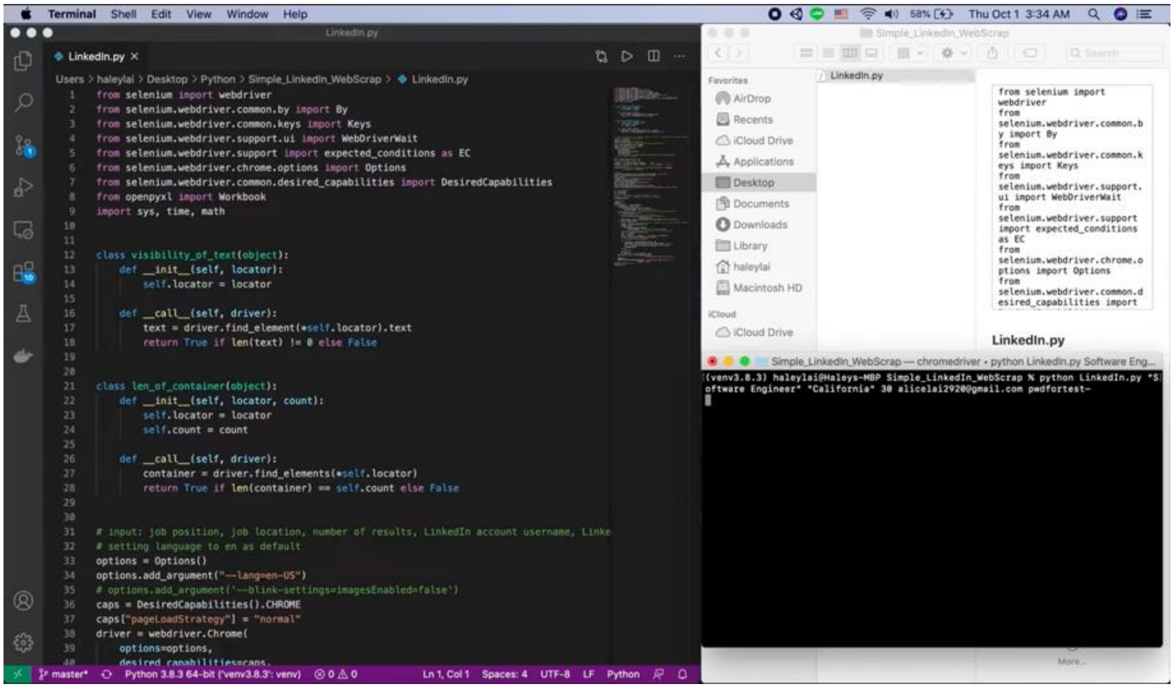

**Zombie attacks human**

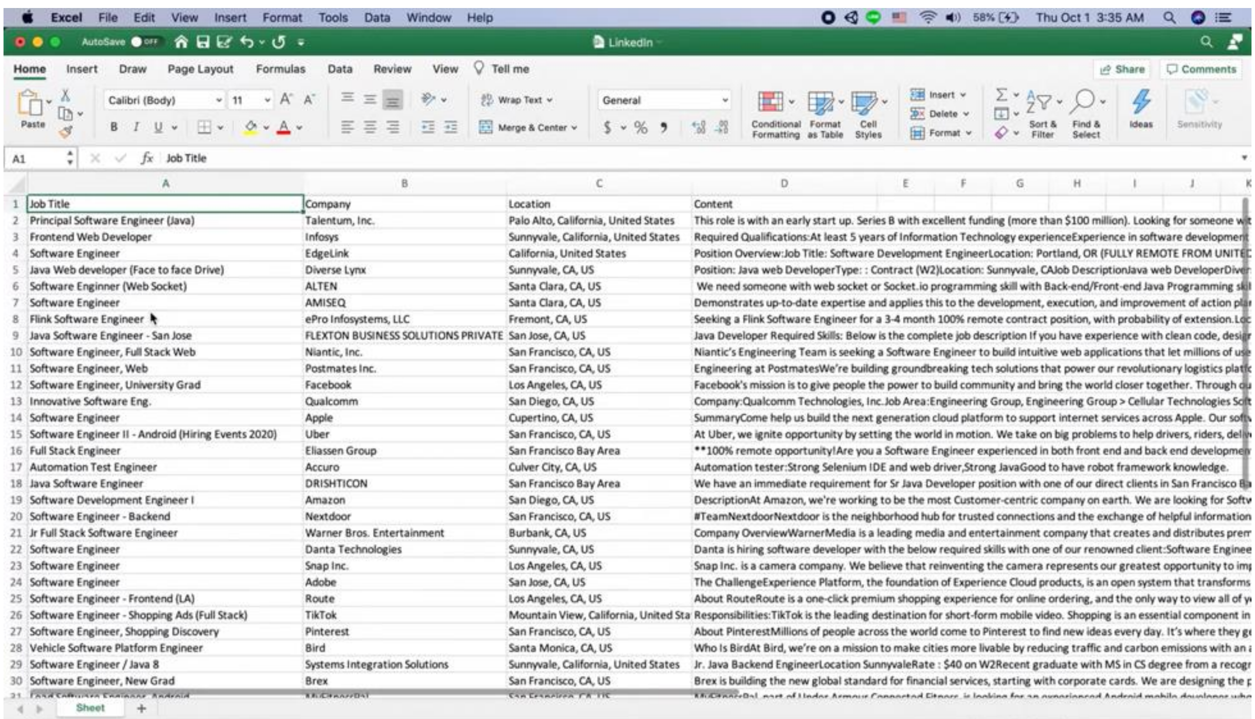

**Zombie gets hurt by the trap**

# Web Scrapper

## LinkedIn Web Scrap (Python, Selenium)

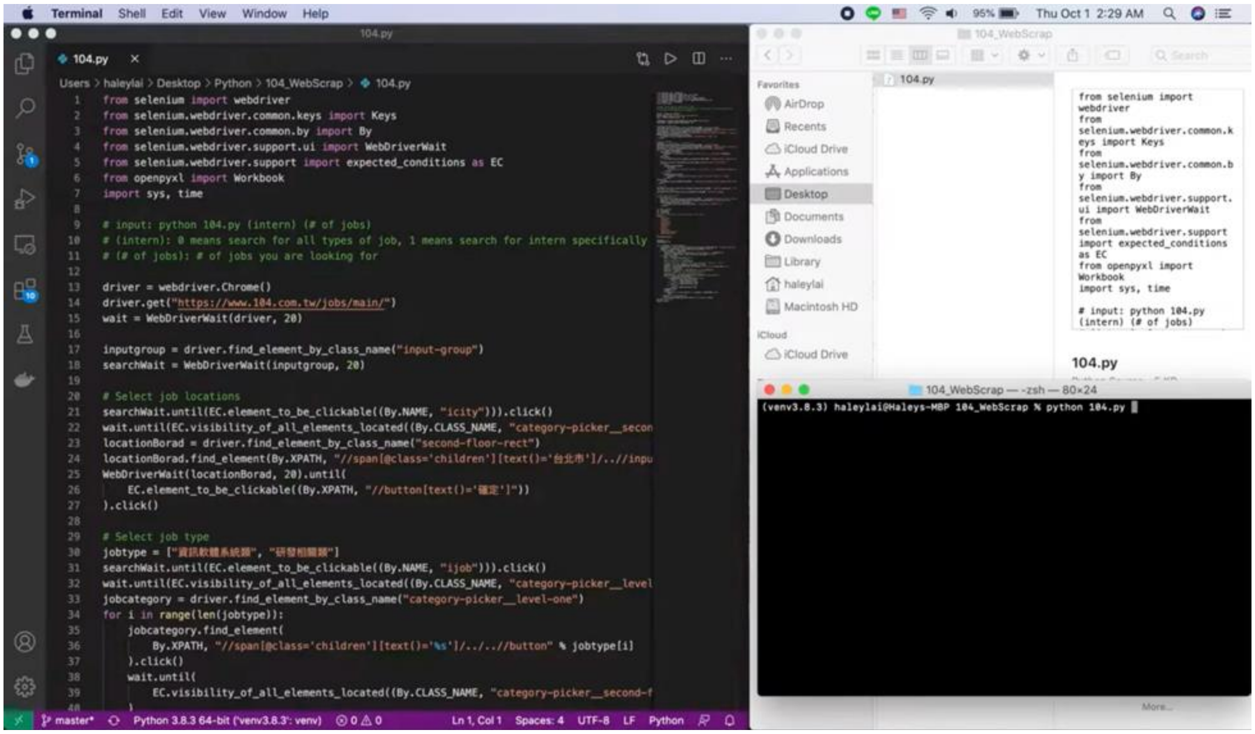
- Scraping job info from LinkedIn, including job title, company, location, and job content

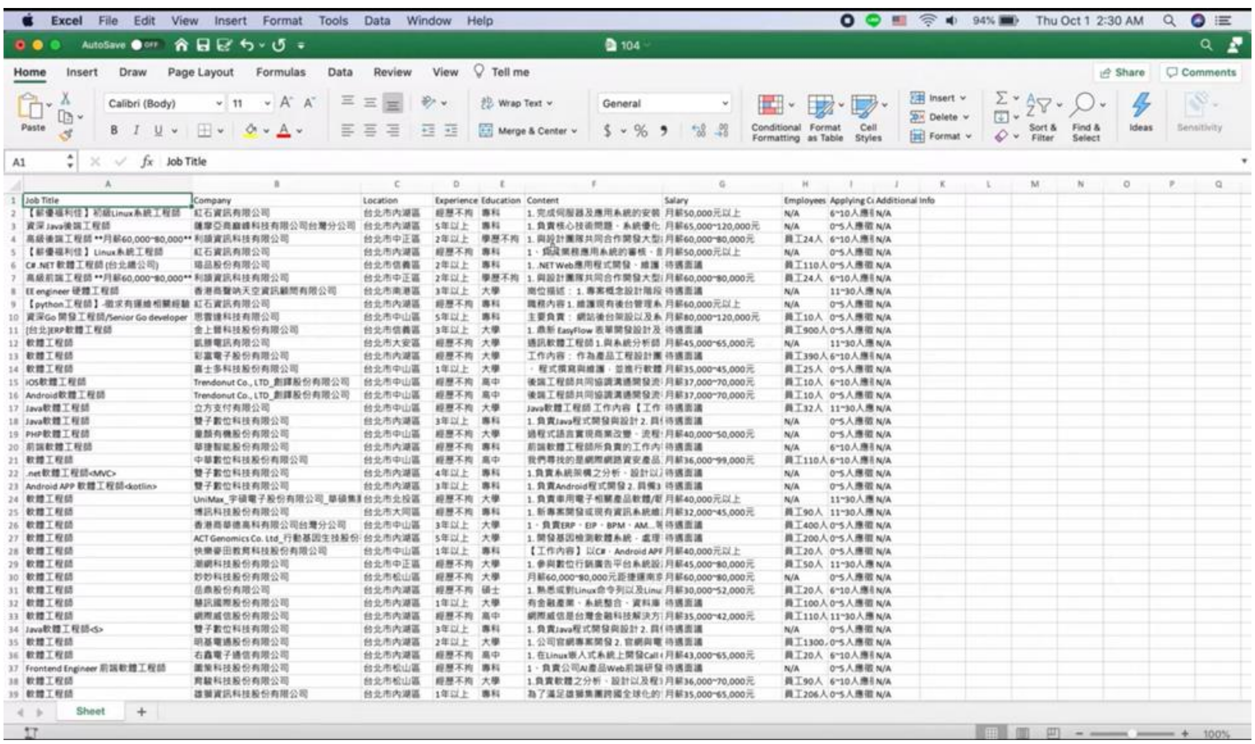


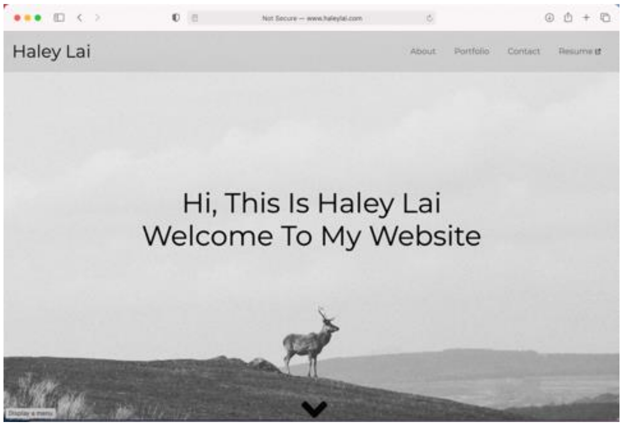
## 104 Web Scrap (Python, Selenium)

- Scrape job info from 104.com, including job title, company, location, experience, content, employees, applying condition, and additional info

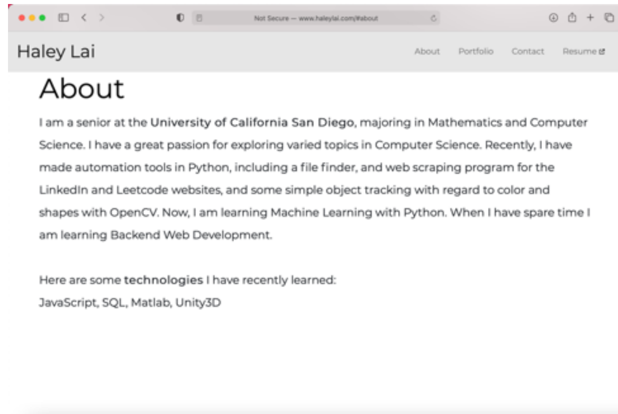


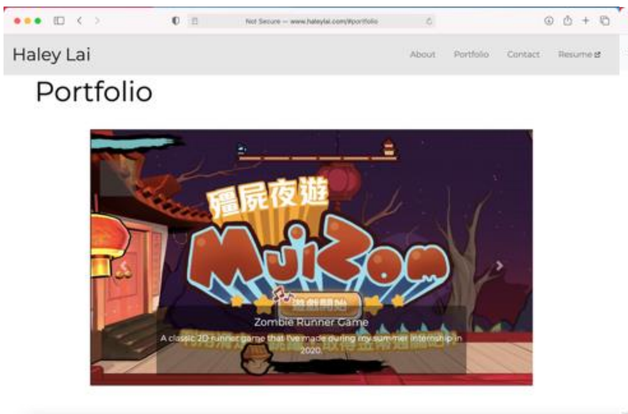
## [Personal Website (HTML, CSS, JavaScript, jQuery, Node.js, Express.js, EJS, MongoDB)]



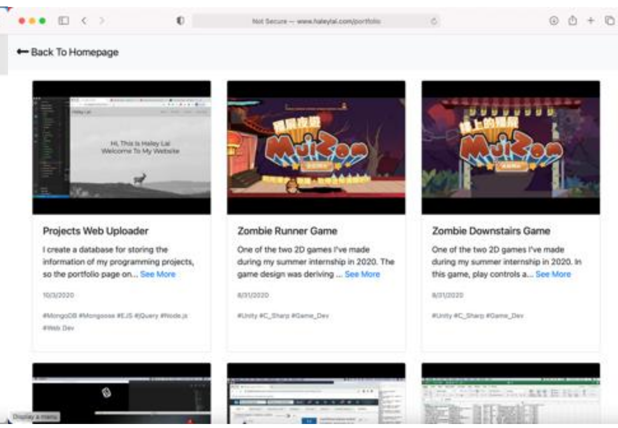

*Home Page*



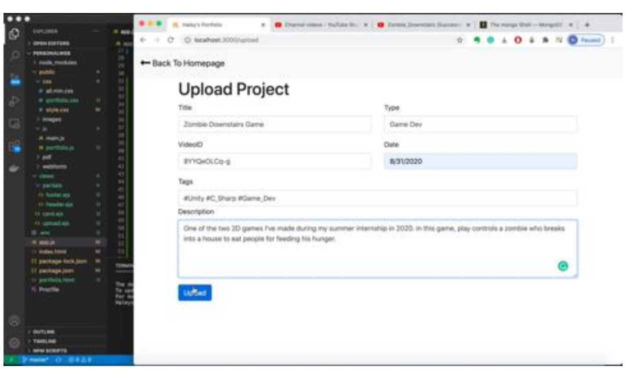

*About Page*



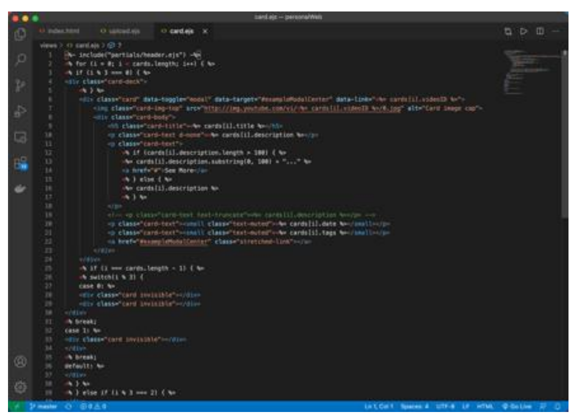

*Portfolio Overview*




*Portfolio Page*




*Portfolio page uploader*



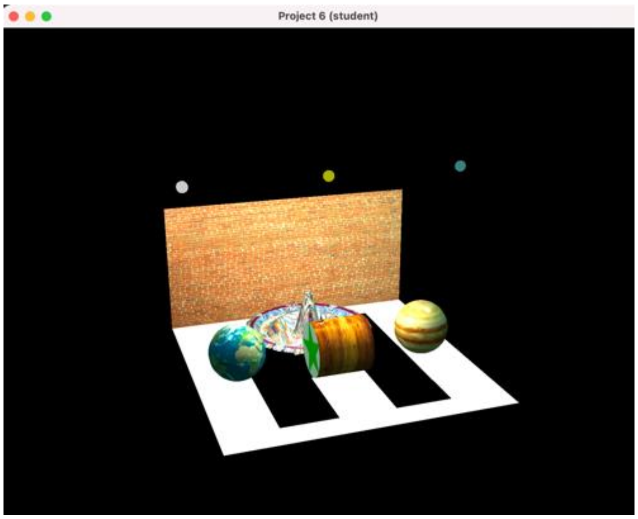
*Partial code of project page uploader*

Portfolio page uploader:
A simple page I made for uploading projects to my portfolio page. A project card (refer to Portfolio Page figure) will be created automatically after submitting the form in the upload page.

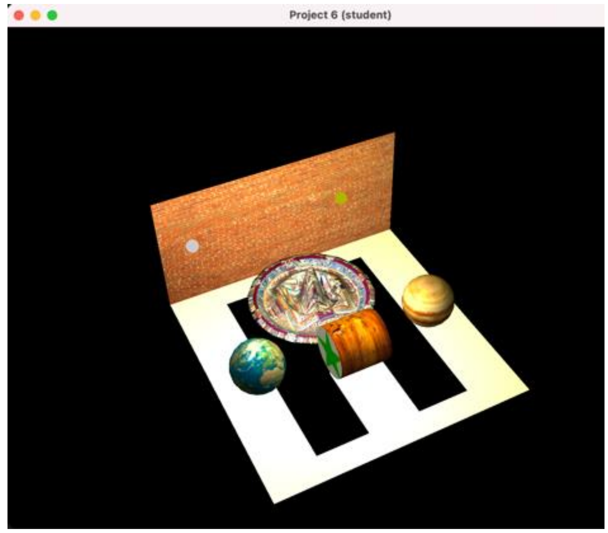
# Computer Graphic

## Texture Maps (C++ with OpenGL)

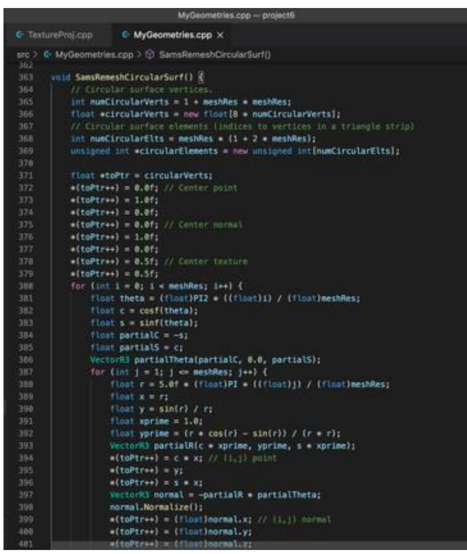
- Add textures to spheres, a rectangle, a cylinder and a surface of rotation.
- Add texture coordinates to the surface of rotation.
- Write a procedural texture for the floor of the scene.



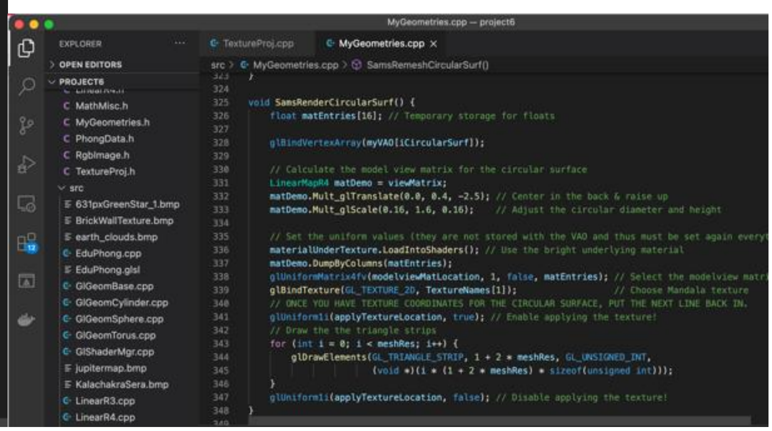

*Objects with text*




*Toggle ambient lighting off*



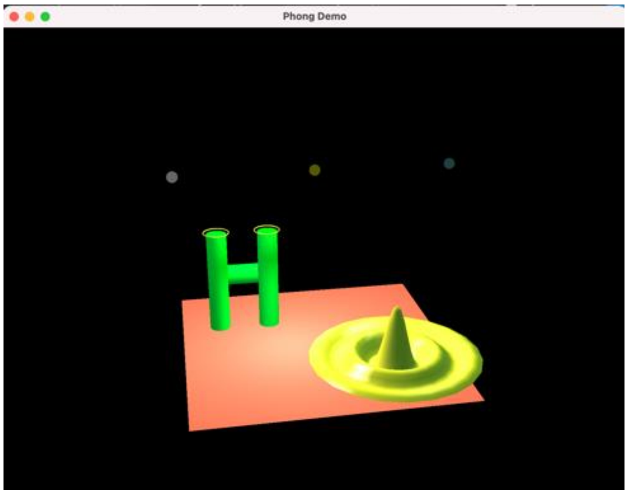

Calculate texture coordinate for circular surface



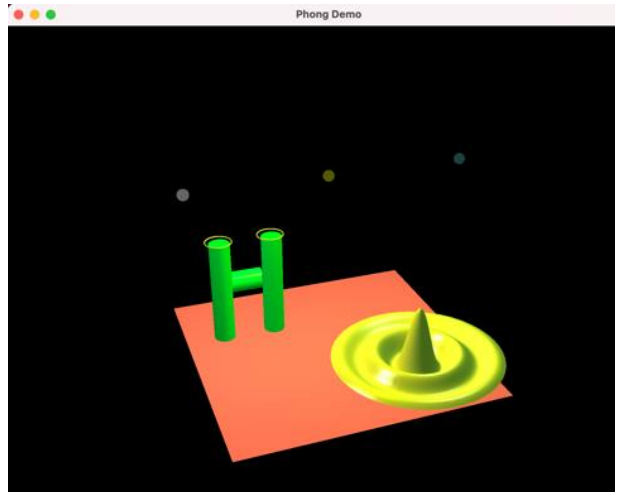

Add texture to circular surface

## Phong Lighting (C++ with OpenGL)

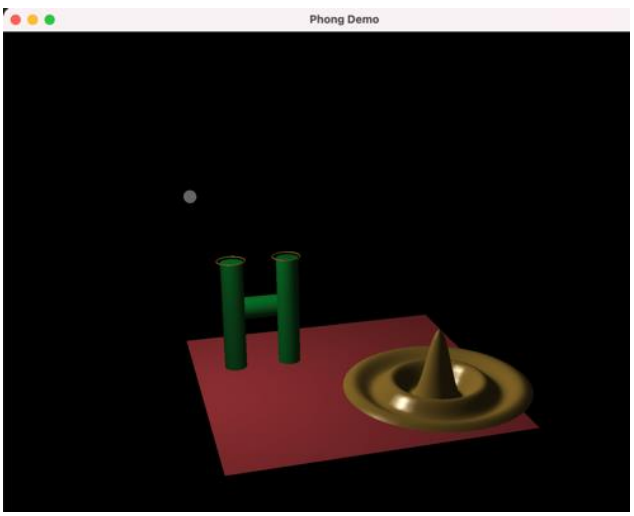
- Use illumination and shading to make scene look more three-dimensional.
- Learn how to shade objects with the Phong lighting model in OpenGL.
- Create four lights, including a spotlight, and create three materials.



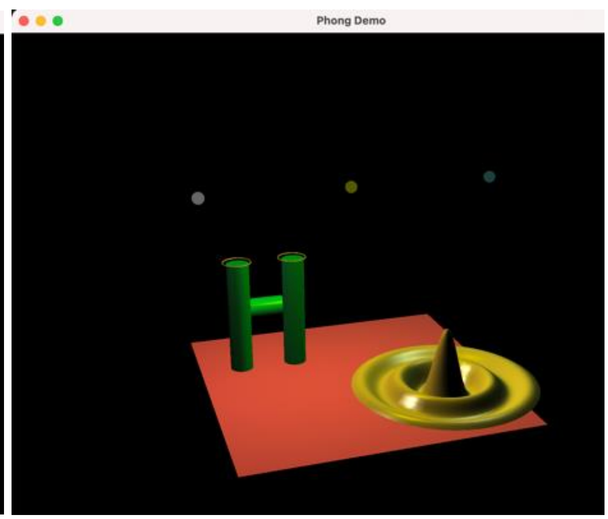
*All lights are on*



*Turn spotlight off*



*Only light1 is on*

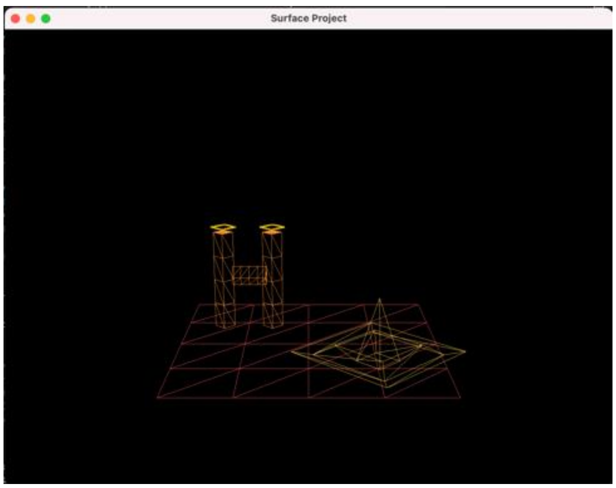


*Toggle ambient lighting off*

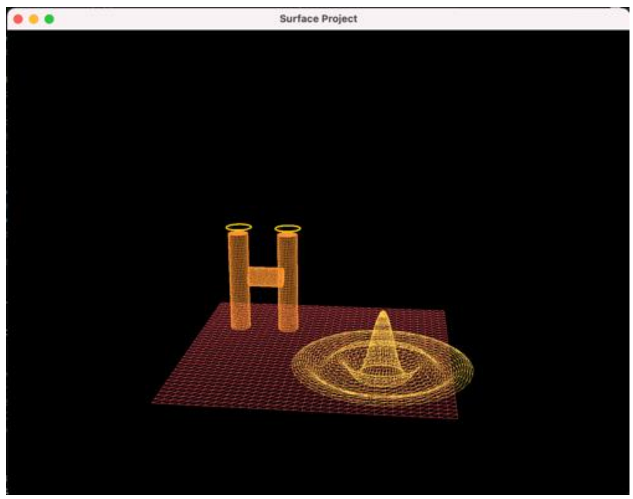
## Surface of Rotation and Normals (C++ with OpenGL)

- Create a parametric surface (namely, a surface of rotation) using triangle strips.
- Create a rectangular mesh for the ground plane using triangle strips.
- Dynamically change the resolution of the surface of rotation and the ground plane.
- Correctly calculate normal for both objects.
- Discover that wireframe objects, especially when combined with animation can look very three-dimensional.
- Discover, however, that flat, solidly colored objects look much too flat and non-three-dimensional.

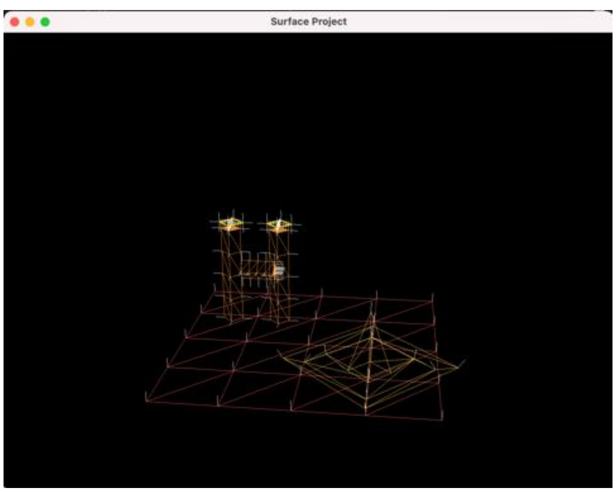
Key press events:
'w': toggle wireframe mode
'c': toggle backfaces culling
'M' or 'm': increases or decreases the mesh resolution
'N' or 'n': visualizing surface normals



*Scene with low resolution*



*Scene with high resolution*



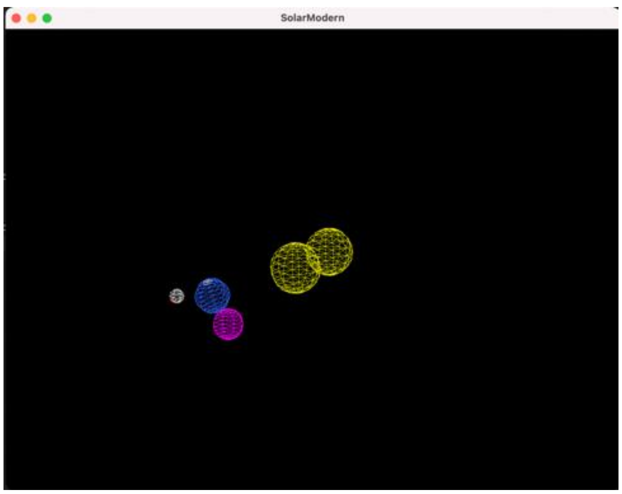
*Visualizing normals under low resolution*

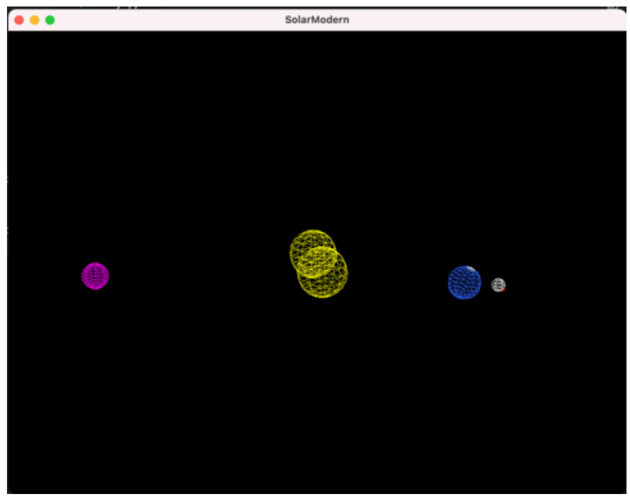
*Visualizing normals under high resolution*

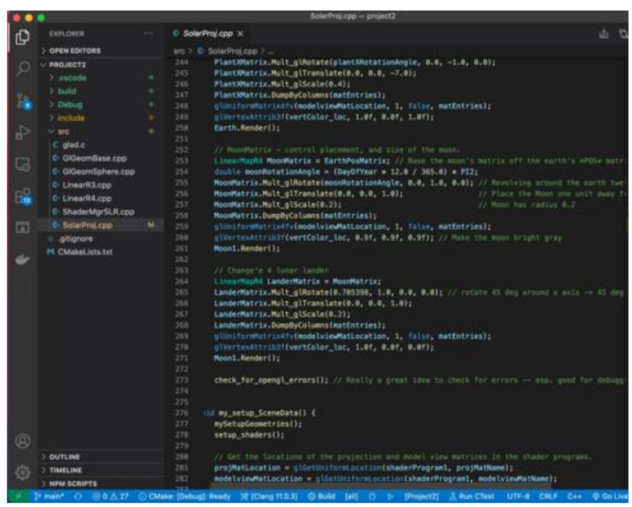
## [Solar System (C++ with OpenGL)]

- Learn more about how to use OpenGL, interrupt-driven programming, animation, and transformations. Program some additions to an animated solar system. Use OpenGL commands to generate transformations that control the animation.

Scene explanation:
**Binary Sun in the middle** (Yellow spheres): The two suns revolve around the center of the solar system.
**Earth** (Blue sphere): revolve around the center of the solar system
**Moon** (White sphere): revolve around the Earth
**Planet X** (Magenta sphere): orbit the Sun once every 600 days (slower than the Earth)
**Change'e 4 Lunar Lander** (Red Dot on the moon): a fixed location on the moon



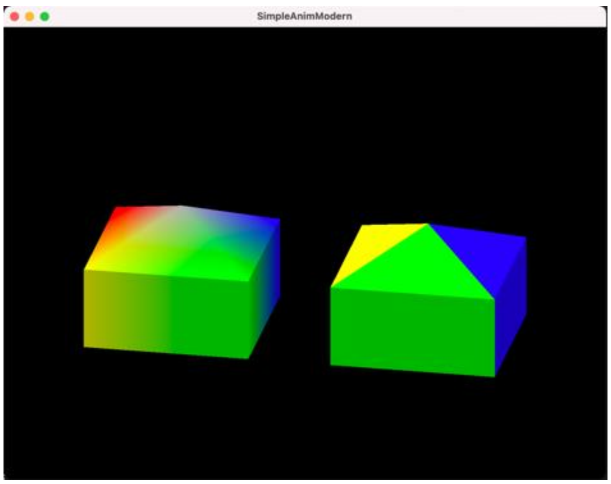




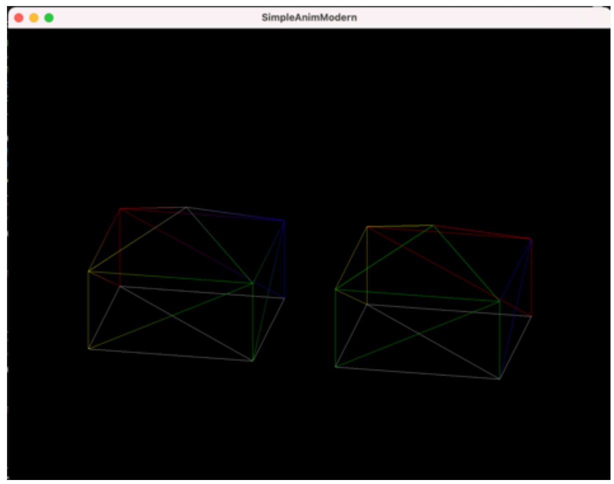
## [Shaded Tent (C++ with OpenGL)]

- Rendering a tent by using 3 different *glDrawArrays* methods, GL_TRIANGLE_FAN, GL_TRIANGLE_STRIP, and GL_TRIANGLES
- Learn how to make triangles of solid color as well as how to shade colors smoothly, how to use key controls to control viewpoint, and toggle wireframe and toggle culling of back faces.

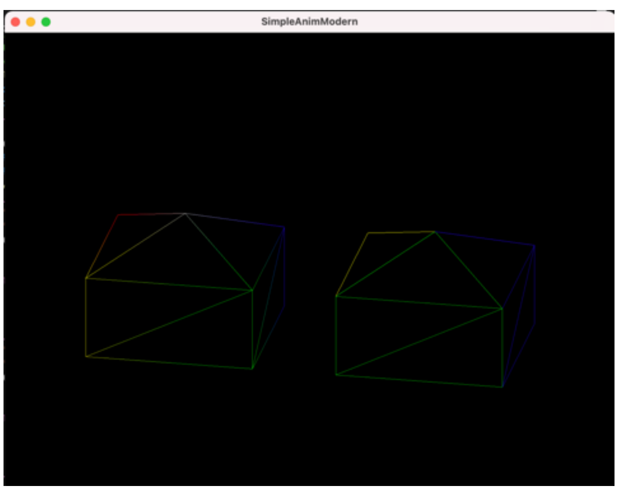
Key press events:
'w': toggle wireframe mode
'c': toggle backface culling



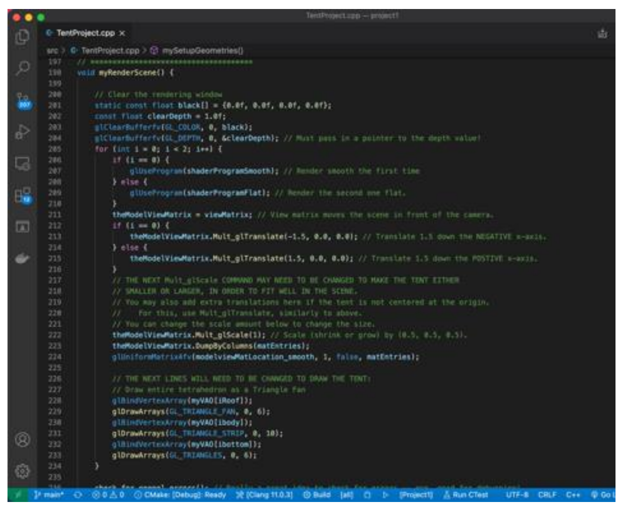

*Result Tent*




*In wireframe mode*




*Culling backface*




*Partial code of rendering scene*

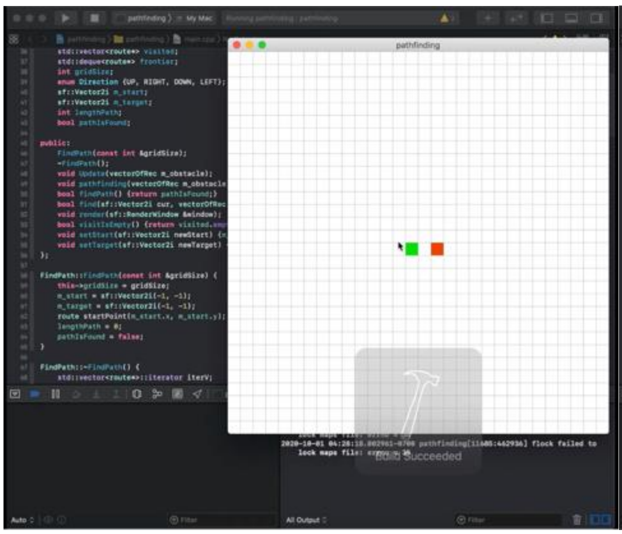
# Early Projects (Focus on Game development and algorithm visualization)

## [Pathfinding Visualizer (C++ with SFML framework)]

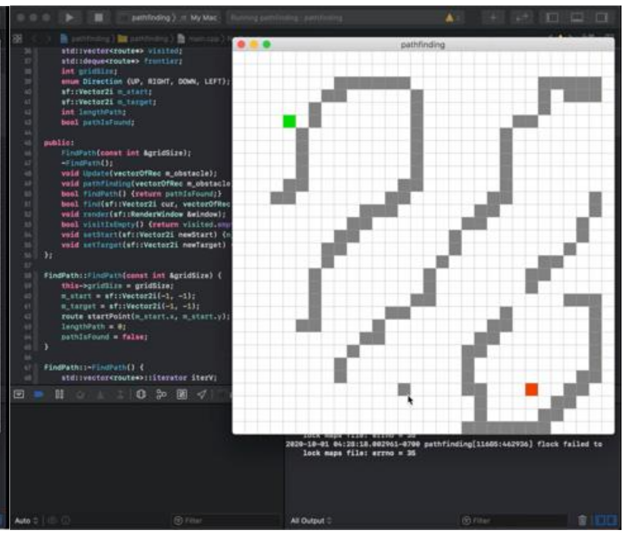
- BFS search algorithm

How to play:

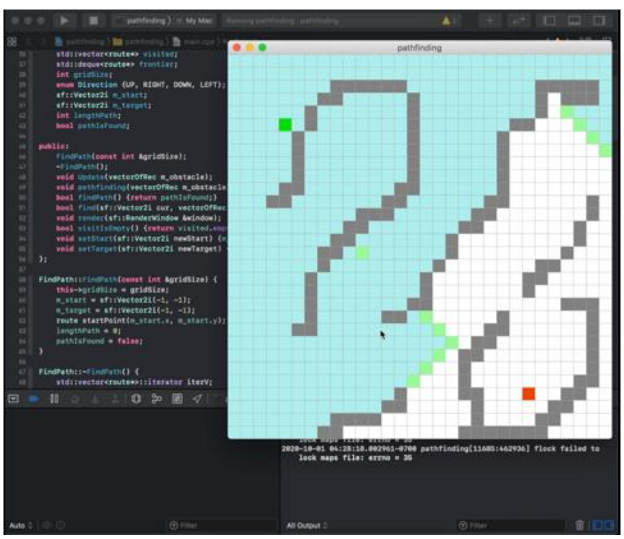
- Drag the green grid and red grid around the board to set the position of the starting point and the target.
- Drawing obstacles
  - o Click anywhere on an empty square (white square) to create an obstacle on the board.
  - o Hold the mouse while drawing obstacles to create continuous obstacles.
- Removing obstacles
  - o Click an obstacle (gray square) again to remove an obstacle on the board.
  - o Users can hold the mouse while removing obstacles to clear quickly.
- Click Enter to run the program to show the path between starting point and the target.



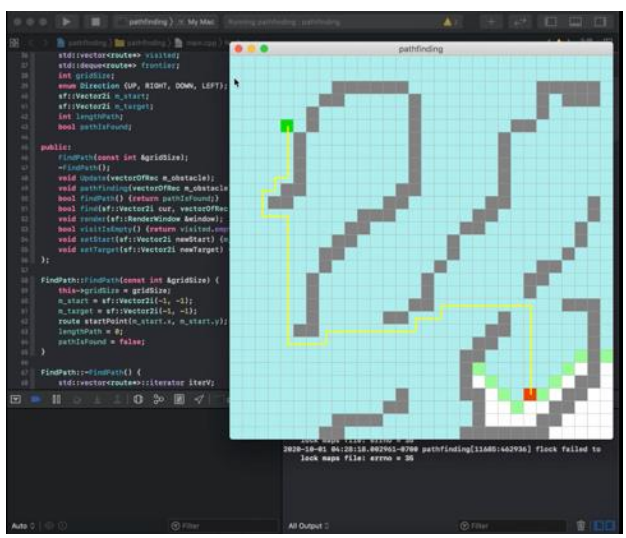
*Start Board*



*Set Obstacles*



*Start finding path*



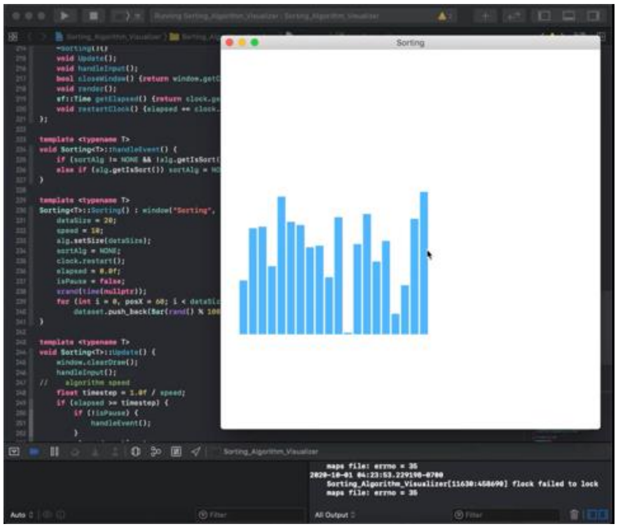
*Find a shortest path*

## Sorting Visualizer (C++ with SFML framework)

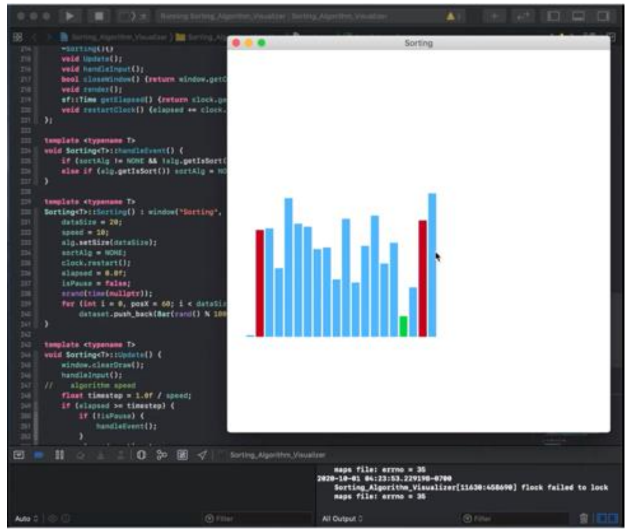
- The following figures are based on selection sort
- Left red bar represents the current index
- Right red bar represents the iterator (loop from current index to the last index to find $i^{th}$ smallest value in the dataset)
- Green bar represents the current $i^{th}$ smallest value

How to play:

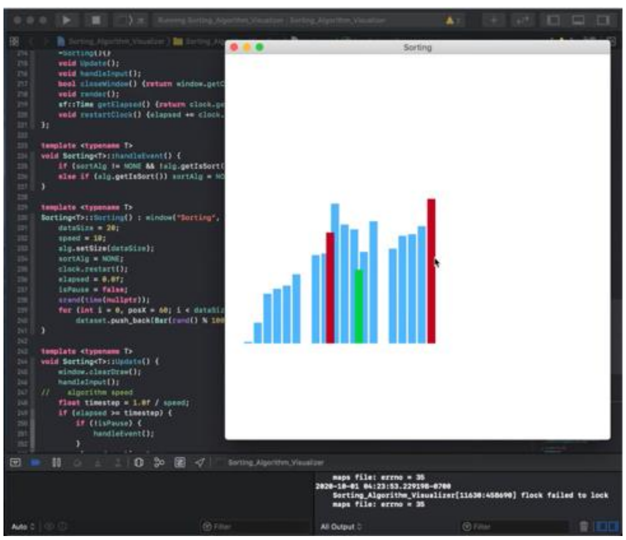
- Press Enter to start sorting



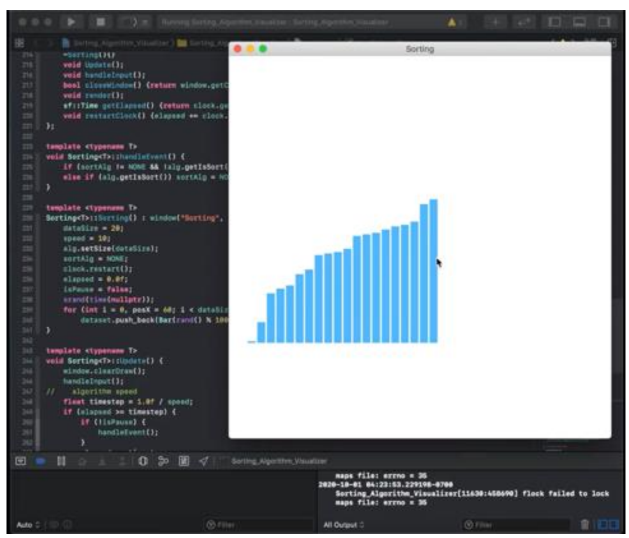
*Start Random Dataset*



*Start sorting*



*Swapping current index with smallest value*



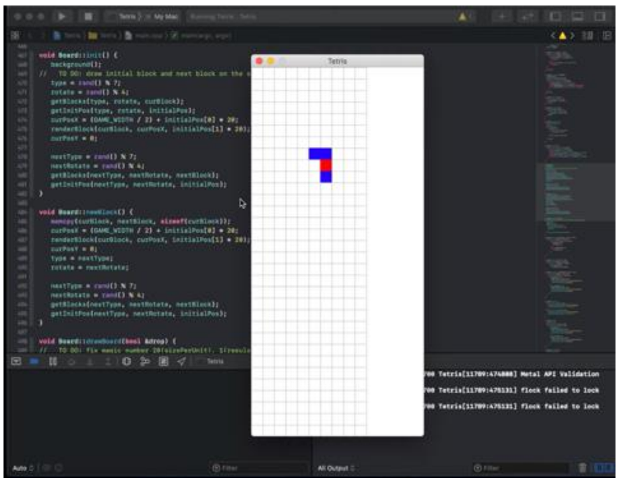
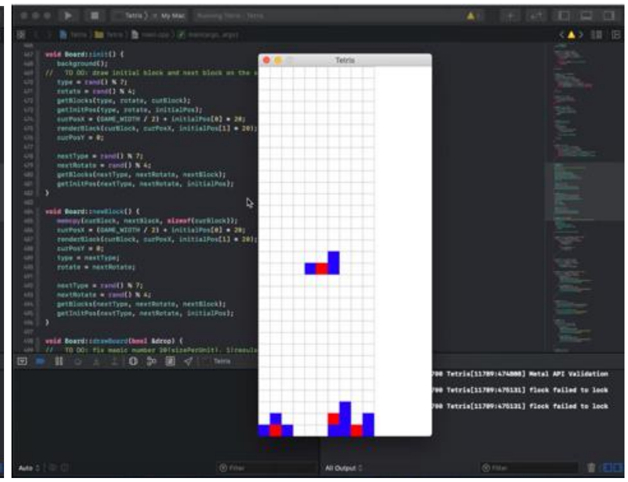
*Done sorting*

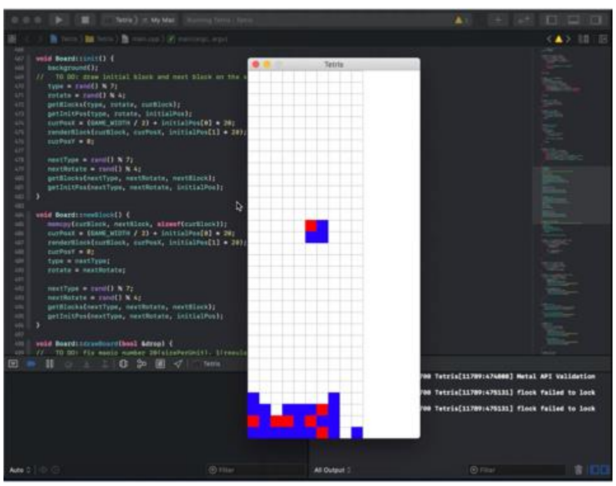
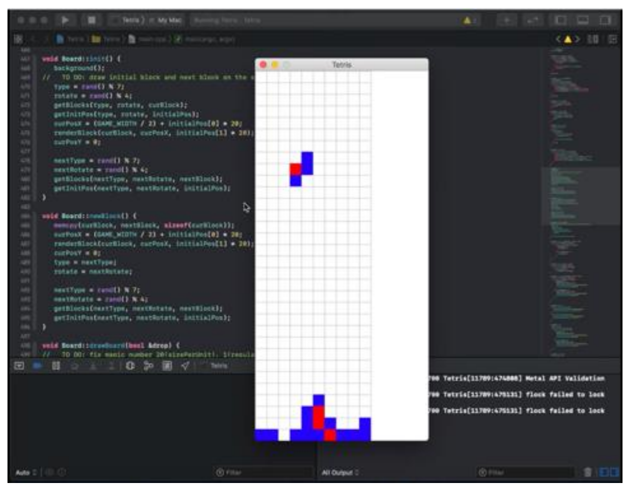
## Tetris (C++ with SDL framework)

- Classic Tetris game made by C++ and SDL framework.
- The red grid represents the pivot in each block

How to play:

- Use the left and right arrow keys to move laterally a falling block.
- Use the down arrow key to accelerate the falling speed.
- Use the z key to rotate a block left and x key to rotate right.

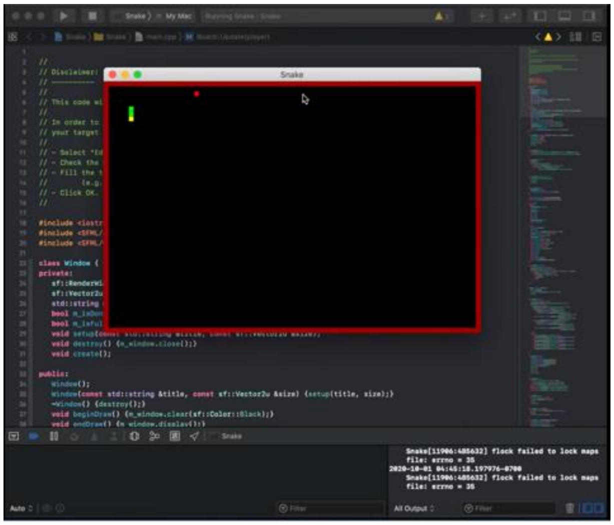



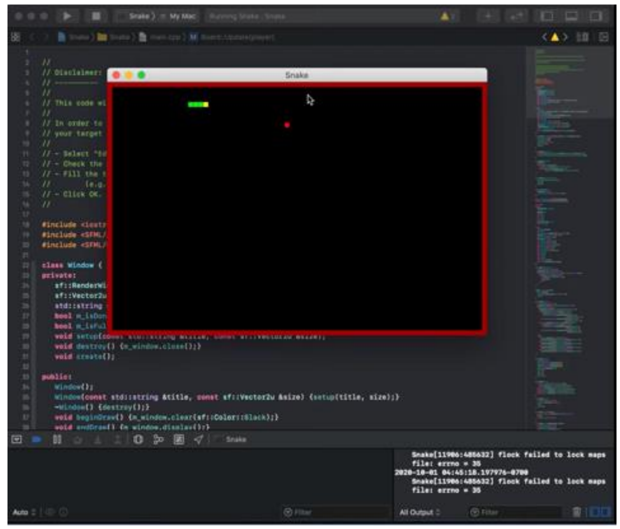
## Snake (C++ with SFML framework)

- A classic snake game made by C++ and SFML framework

How to play:

- Use arrow keys to move the snake within the board.
- Be careful not to hit the red barriers.
- Try to eat as many as apple (red dot) on the board to extend the body.






## Breakout (C++ with SDL framework)

- Breakout game made by C++ and SFML framework.

How to play:

- Use the left arrow key and right arrow key to move the paddle to ricochet the ball.
- After knocking down all the bricks in the game, the player wins