Imports for machine learning algorithms and plots

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import sys
!sudo {sys.executable} -m pip install sklearn

# data analysis and wrangling
import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

#from mlxtend.frequent_patterns import apriori
```

```
Requirement already satisfied: sklearn in /usr/local/lib/python3.4/dist-package
s
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.4/dist-pa
ckages (from sklearn)
You are using pip version 9.0.1, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Data curation

In [3]:
```
#this is a non functioning example of our data curation. All data curation was do
# have was too large for codio. The below 3 cells contain our data manipulations.
csgo = pd.read_csv('dust2.csv')
de_dust2 = csgo[csgo['map'] == "de_dust2"]
de_dust2.loc[:,'victim'] = False
de_dust2.loc[:,'nade_land'] = False
```

In [4]:
```
values = {'vic_pos_x' : 0, 'vic_pos_y' : 0, 'vic_side' : 'Terrorist'} #this isnt
de_dust2 = de_dust2.fillna(value=values)
```

In [5]:
```
for i,row in de_dust2.iterrows():
    if row['vic_pos_x'] != 0 and row['vic_pos_y'] != 0:
        de_dust2.set_value(i,'victim',True)
    if row['nade_land_x'] != 0 and row['nade_land_y'] != 0:
        de_dust2.set_value(i,'nade_land',True)
```

In [6]:
```
dust_data = pd.read_csv("dust2.csv")
dust_data.head()
```

Out[6]:

	Unnamed: 0	Unnamed: 0.1	file	map	round	start_
0	0	0	003201673717864202280_0171883906.dem	de_dust2	1	109.5
1	1	1	003201673717864202280_0171883906.dem	de_dust2	1	109.5
2	2	2	003201673717864202280_0171883906.dem	de_dust2	1	109.5
3	3	3	003201673717864202280_0171883906.dem	de_dust2	2	197.6
4	4	4	003201673717864202280_0171883906.dem	de_dust2	2	197.6

5 rows × 35 columns

In [7]:
```
a = list(dust_data.columns.values)
print(a)
print()
nades = dust_data['nade'].unique()
print(nades)
```

```
['Unnamed: 0', 'Unnamed: 0.1', 'file', 'map', 'round', 'start_seconds', 'second
s', 'end_seconds', 'att_team', 'vic_team', 'att_id', 'vic_id', 'att_side', 'vic
_side', 'hp_dmg', 'arm_dmg', 'is_bomb_planted', 'bomb_site', 'hitbox', 'nade',
'winner_team', 'winner_side', 'att_rank', 'vic_rank', 'att_pos_x', 'att_pos_y',
'nade_land_x', 'nade_land_y', 'vic_pos_x', 'vic_pos_y', 'round_type', 'ct_eq_va
l', 't_eq_val', 'avg_match_rank', 'victim']

['Decoy' 'HE' 'Smoke' 'Flash' 'Molotov' 'Incendiary']
```

In [8]:
```
dust_data.shape
```

Out[8]: (85742, 35)

In [9]:
```
dust_data.loc[:,'nade_type'] = 0
for i,row in dust_data.iterrows():
    if row['nade'] == 'Flash':
        dust_data.set_value(i,'nade_type',0)
    elif row['nade'] == 'Smoke':
        dust_data.set_value(i,'nade_type',1)
    elif row['nade'] == 'HE':
        dust_data.set_value(i,'nade_type',2)
    elif row['nade'] == "Incendiary":
        dust_data.set_value(i,'nade_type',3)
    elif row['nade'] == "Molotov":
        dust_data.set_value(i,'nade_type',4)
    elif row['nade'] == "Decoy":
        dust_data.set_value(i,'nade_type',5)
```

In [10]:
```
dust_data.head()
```

Out[10]:

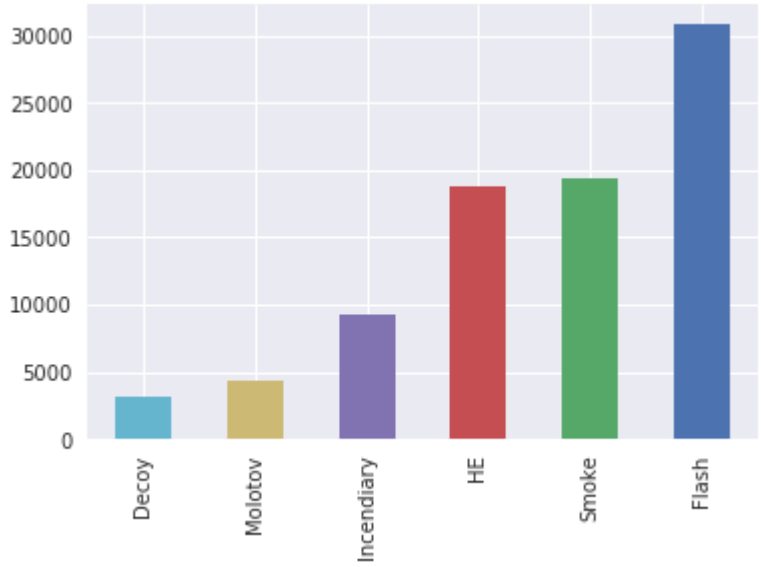
	Unnamed: 0	Unnamed: 0.1	file	map	round	start_
0	0	0	003201673717864202280_0171883906.dem	de_dust2	1	109.5
1	1	1	003201673717864202280_0171883906.dem	de_dust2	1	109.5
2	2	2	003201673717864202280_0171883906.dem	de_dust2	1	109.5
3	3	3	003201673717864202280_0171883906.dem	de_dust2	2	197.6
4	4	4	003201673717864202280_0171883906.dem	de_dust2	2	197.6

5 rows × 36 columns

Explore grenade data with plots:

Frequency of grenade usage:

In [11]:
```
dust_data['nade'].value_counts().plot('bar').invert_xaxis()
```



In [12]:
```
print(dust_data.groupby('nade').size())
```

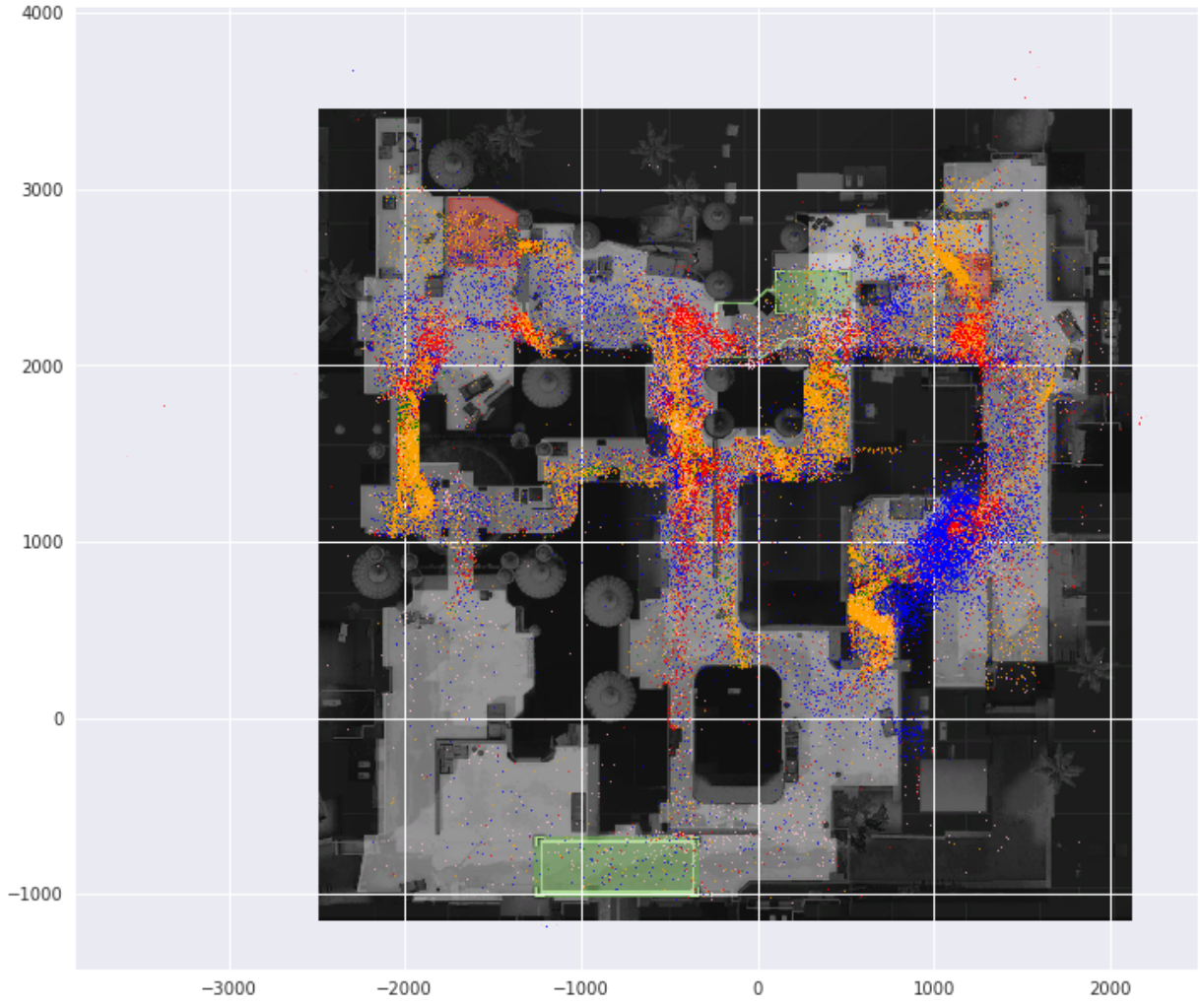
```
nade
Decoy          3207
Flash         30811
HE            18811
Incendiary     9209
Molotov        4310
Smoke         19394
dtype: int64
```

Plot each grenade by type

In [13]:
```
dfFlash = dust_data.loc[dust_data['nade'] == 'Flash']
dfSmoke = dust_data.loc[dust_data['nade'] == 'Smoke']
dfIncen = dust_data.loc[dust_data['nade'] == 'Incendiary']
dfDecoy = dust_data.loc[dust_data['nade'] == 'Decoy']
dfHE = dust_data.loc[dust_data['nade'] == 'HE']
dfMolotov = dust_data.loc[dust_data['nade'] == 'Molotov']
```

In [14]:
```
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
ax.legend()
plt.scatter(dfFlash['nade_land_x'], dfFlash['nade_land_y'], s=0.4, c ='b')
plt.scatter(dfSmoke['nade_land_x'], dfSmoke['nade_land_y'], s=0.4, c ='r')
plt.scatter(dfIncen['nade_land_x'], dfIncen['nade_land_y'], s=0.4, c ='g')
plt.scatter(dfMolotov['nade_land_x'], dfMolotov['nade_land_y'], s=0.4, c ='orange
plt.scatter(dfHE['nade_land_x'], dfHE['nade_land_y'], s=0.4, c ='orange')
plt.scatter(dfDecoy['nade_land_x'], dfDecoy['nade_land_y'], s=0.4, c ='pink')
```

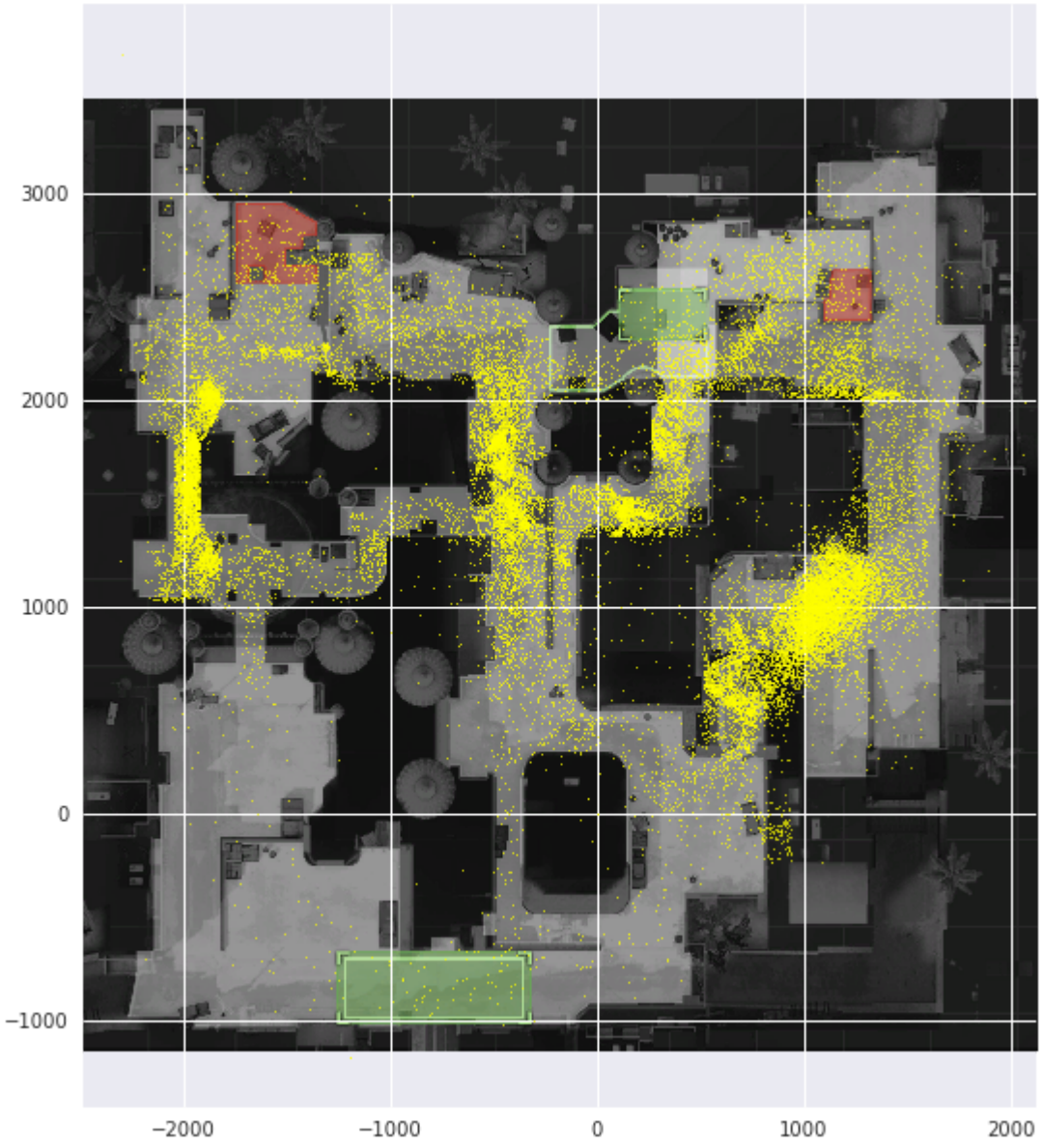
Out[14]: <matplotlib.collections.PathCollection at 0x7f949da4d5c0>



Flash grenade plot:

In [15]:
```
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
ax.legend()
plt.scatter(dfFlash['nade_land_x'], dfFlash['nade_land_y'], s=0.4, c ='yellow')
```

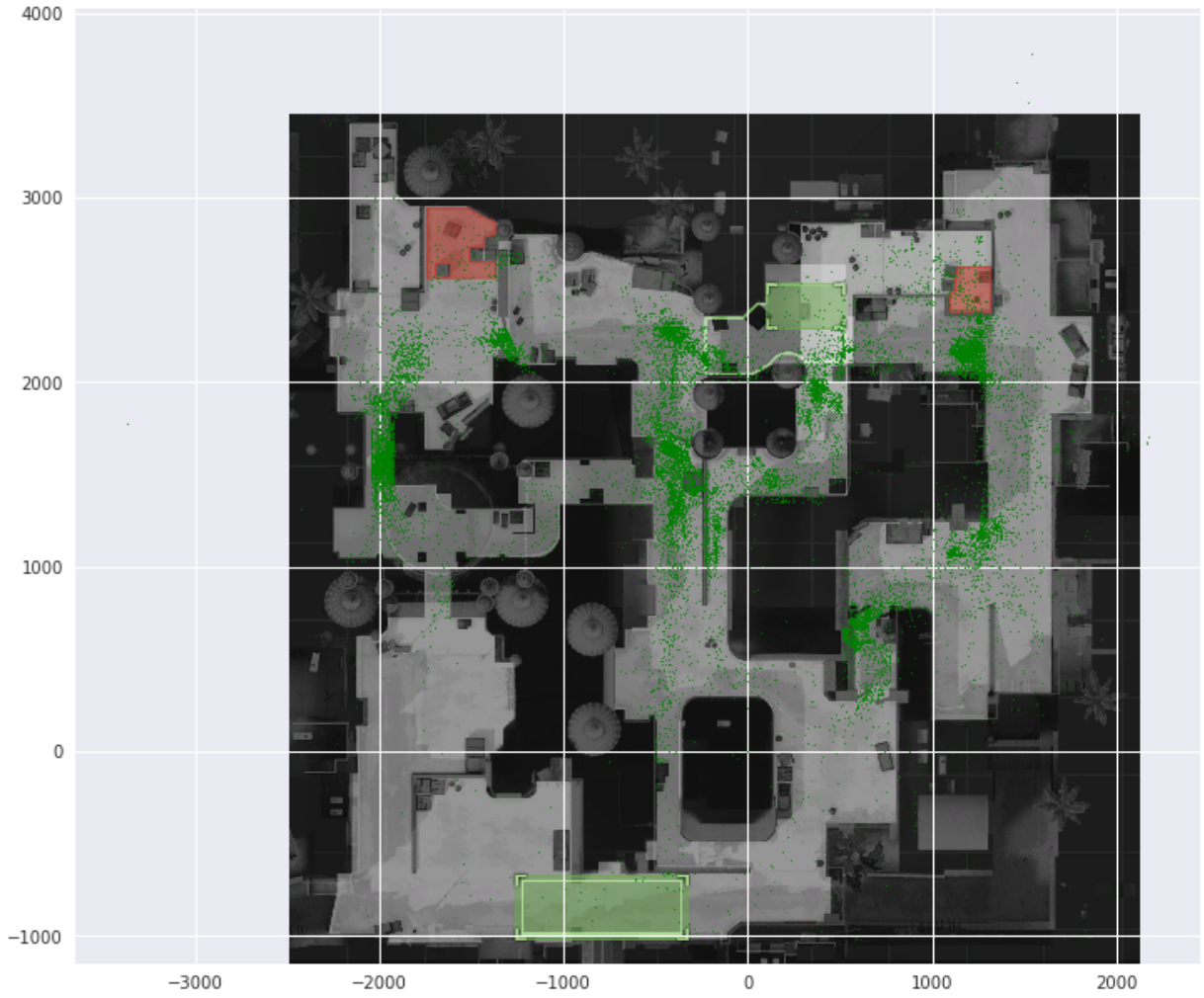
Out[15]: <matplotlib.collections.PathCollection at 0x7f9498f45ef0>



Smoke grenade plot:

In [16]:
```
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
ax.legend()
plt.scatter(dfSmoke['nade_land_x'], dfSmoke['nade_land_y'], s=0.4, c ='g')
```

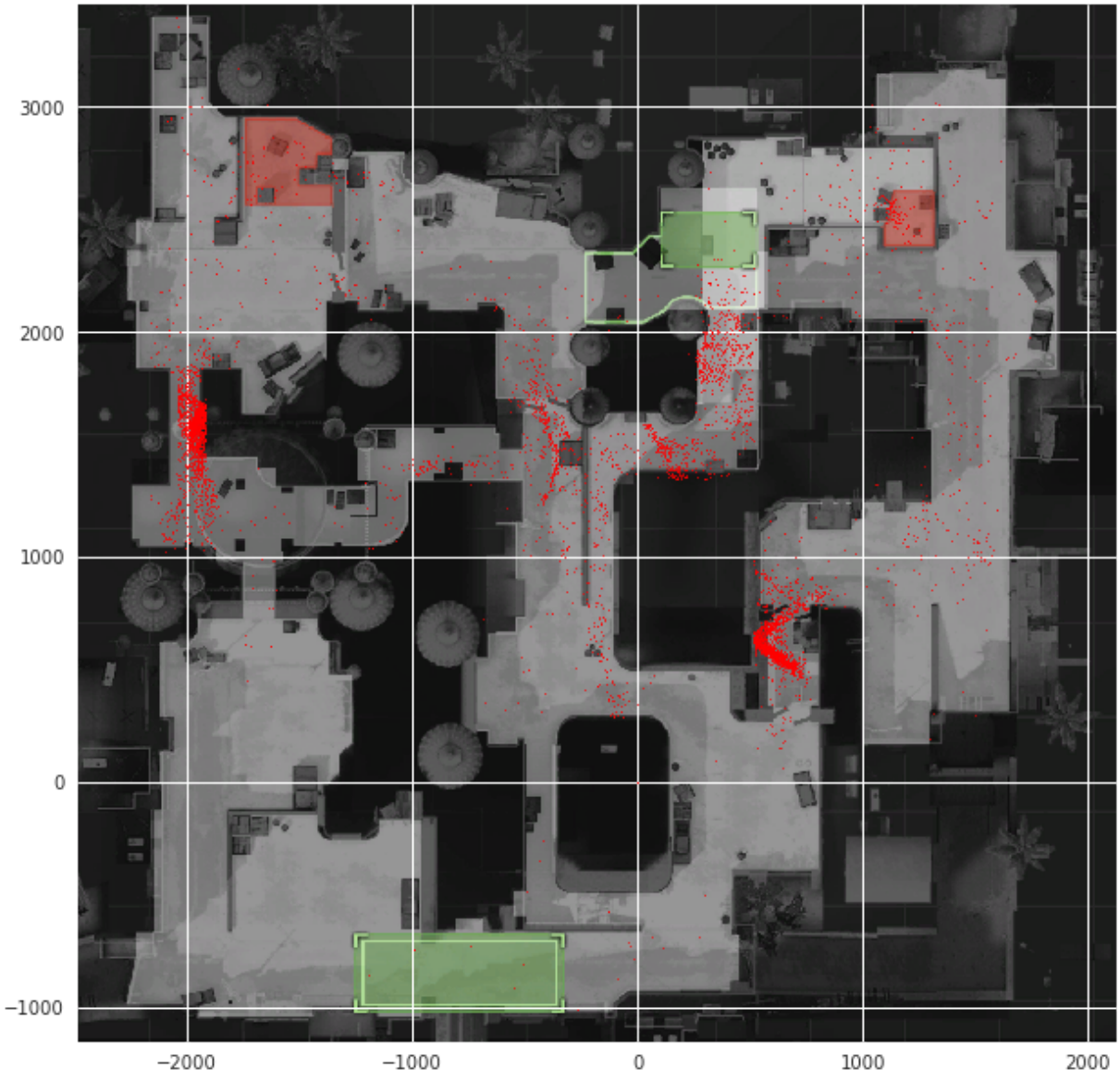
Out[16]: <matplotlib.collections.PathCollection at 0x7f949d9eda58>



Incendiary grenade plot:

In [17]:
```
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
ax.legend()
plt.scatter(dfIncen['nade_land_x'], dfIncen['nade_land_y'], s=0.4, c ='r')
```

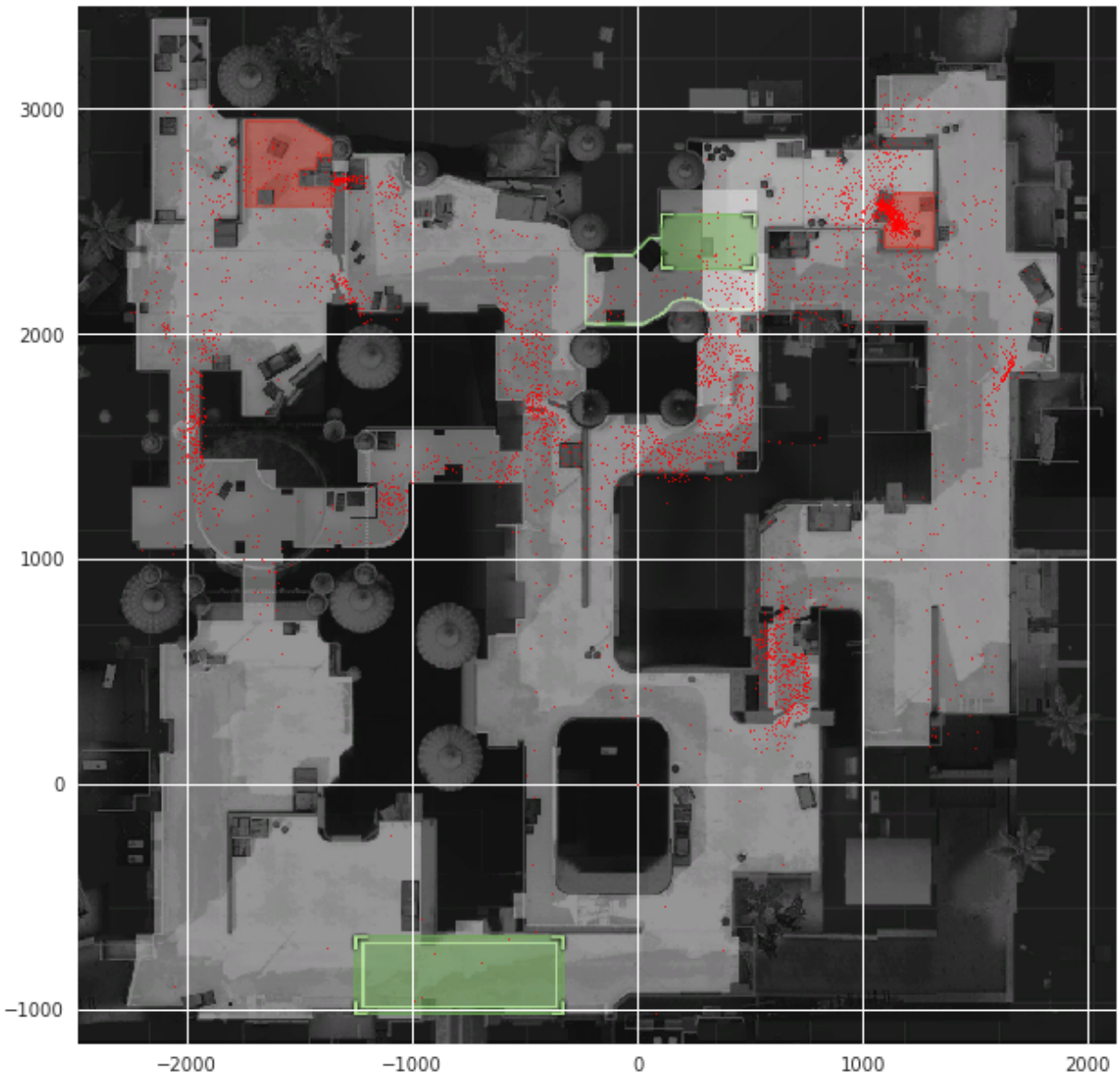
Out[17]: <matplotlib.collections.PathCollection at 0x7f949c5c2048>



Molotov grenade plot:

In [18]:
```
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
ax.legend()
plt.scatter(dfMolotov['nade_land_x'], dfMolotov['nade_land_y'], s=0.4, c ='r')
```

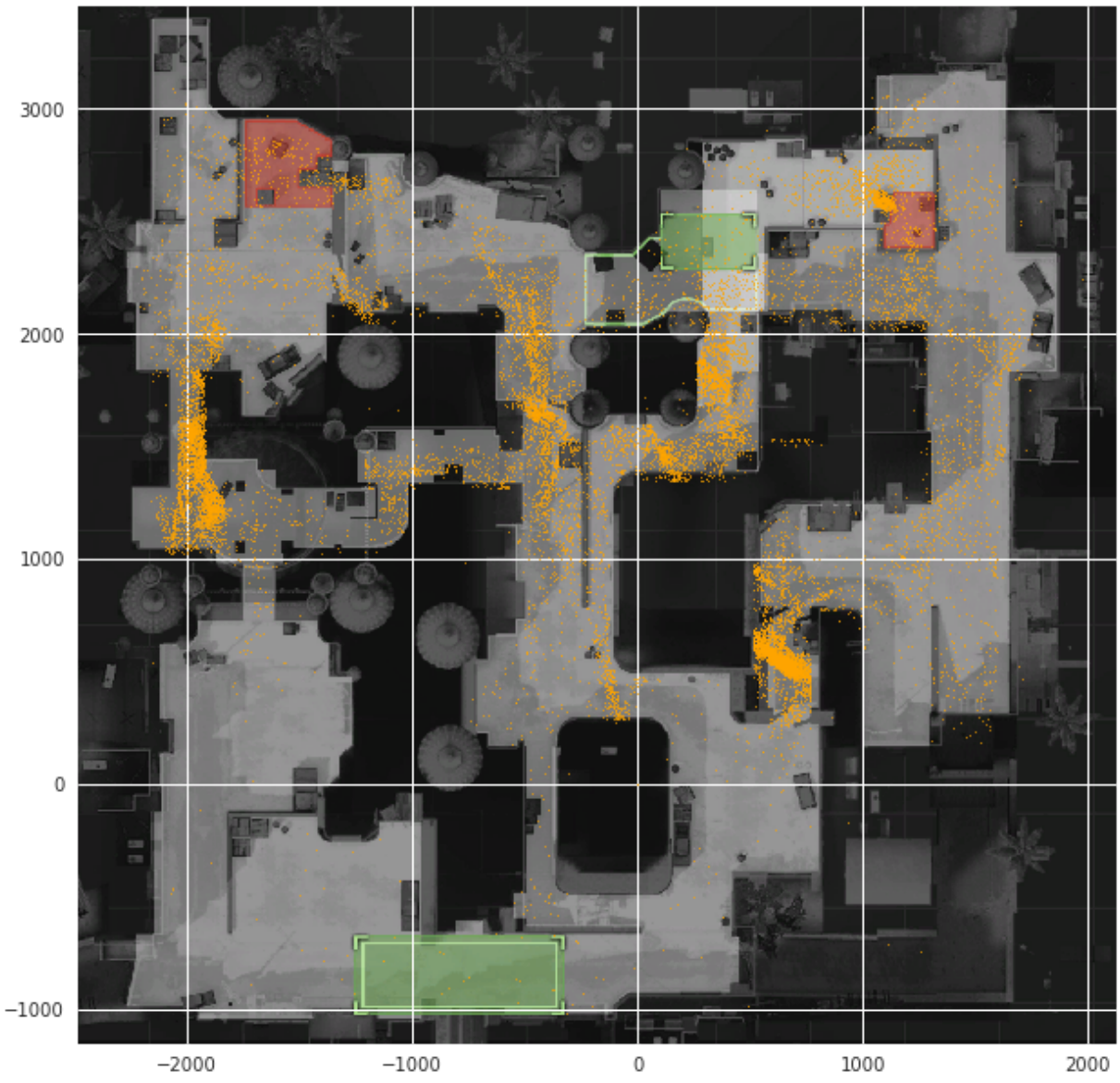
Out[18]: <matplotlib.collections.PathCollection at 0x7f949d997ba8>



High Explosive grenade plot:

In [19]:
```
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
ax.legend()
plt.scatter(dfHE['nade_land_x'], dfHE['nade_land_y'], s=0.4, c ='orange')
```

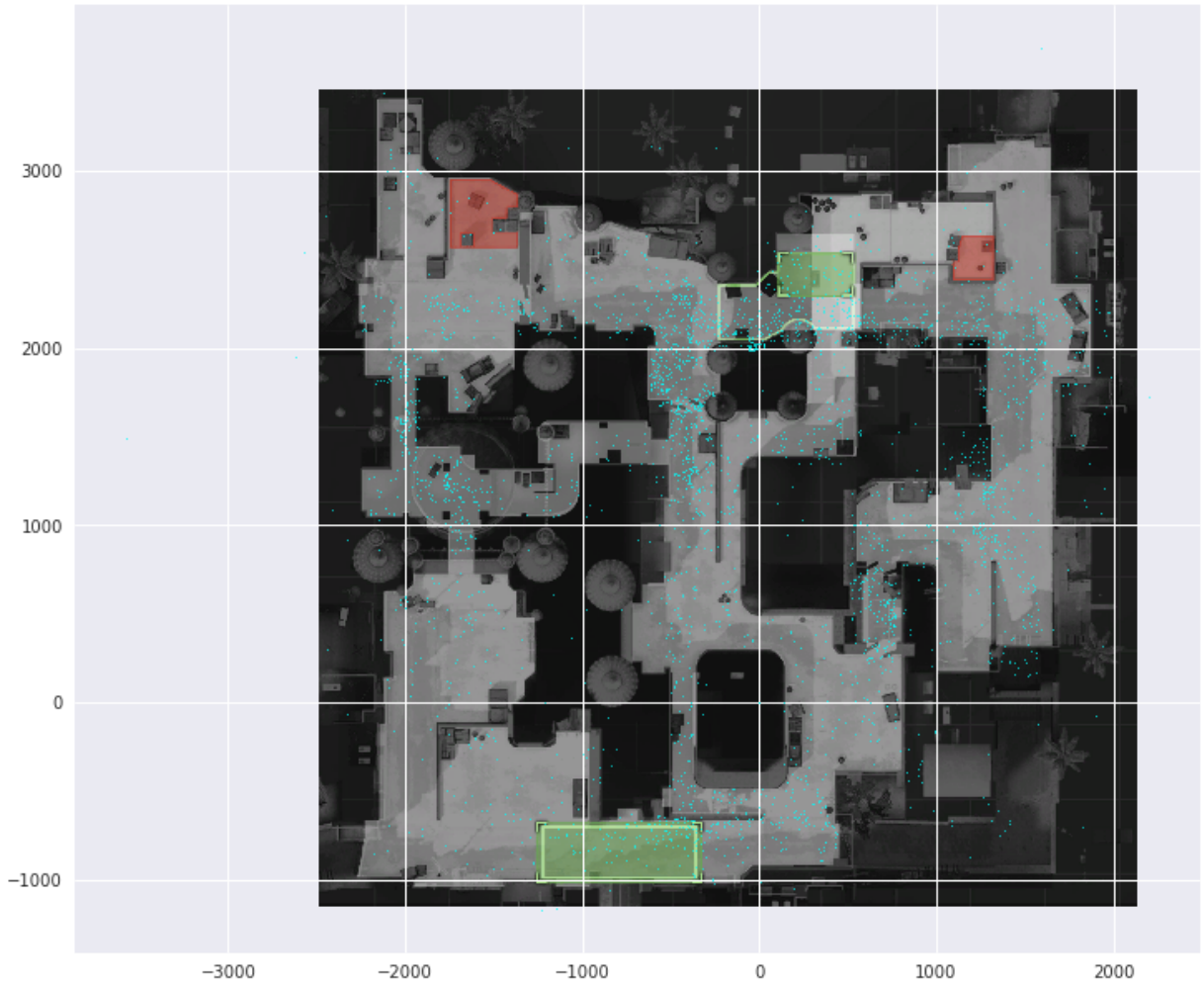
Out[19]: <matplotlib.collections.PathCollection at 0x7f949d95c748>



Decoy grenade plot:

In [20]:
```
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
ax.legend()
plt.scatter(dfDecoy['nade_land_x'], dfDecoy['nade_land_y'], s=0.4, c ='cyan')
```

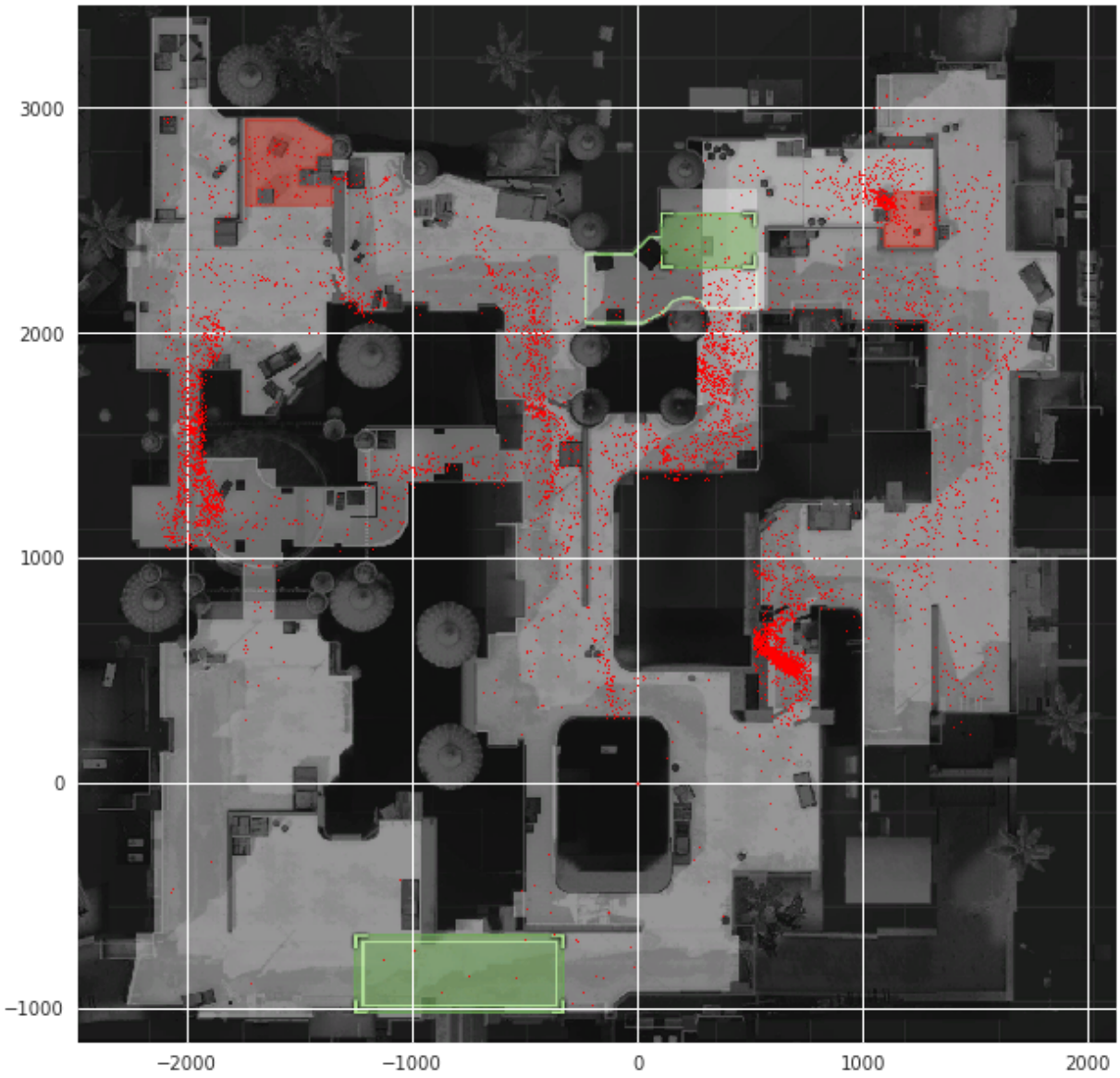
Out[20]: <matplotlib.collections.PathCollection at 0x7f949d947358>



Grenade plot where there was a victim:

In [21]:
```
#other things to add: explore damaging (incendiary, molotov, HE) grenades and find
#where people are most likely to be hit by damaging grenades

#use victim instead of hp_dmg because victim includes flashes while hp_dmg and ar
hitsOnly = dust_data.loc[dust_data['victim'] == True]

#plot damaging grenades
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
plt.scatter(hitsOnly['nade_land_x'], hitsOnly['nade_land_y'], s=0.5, c ='r')
```

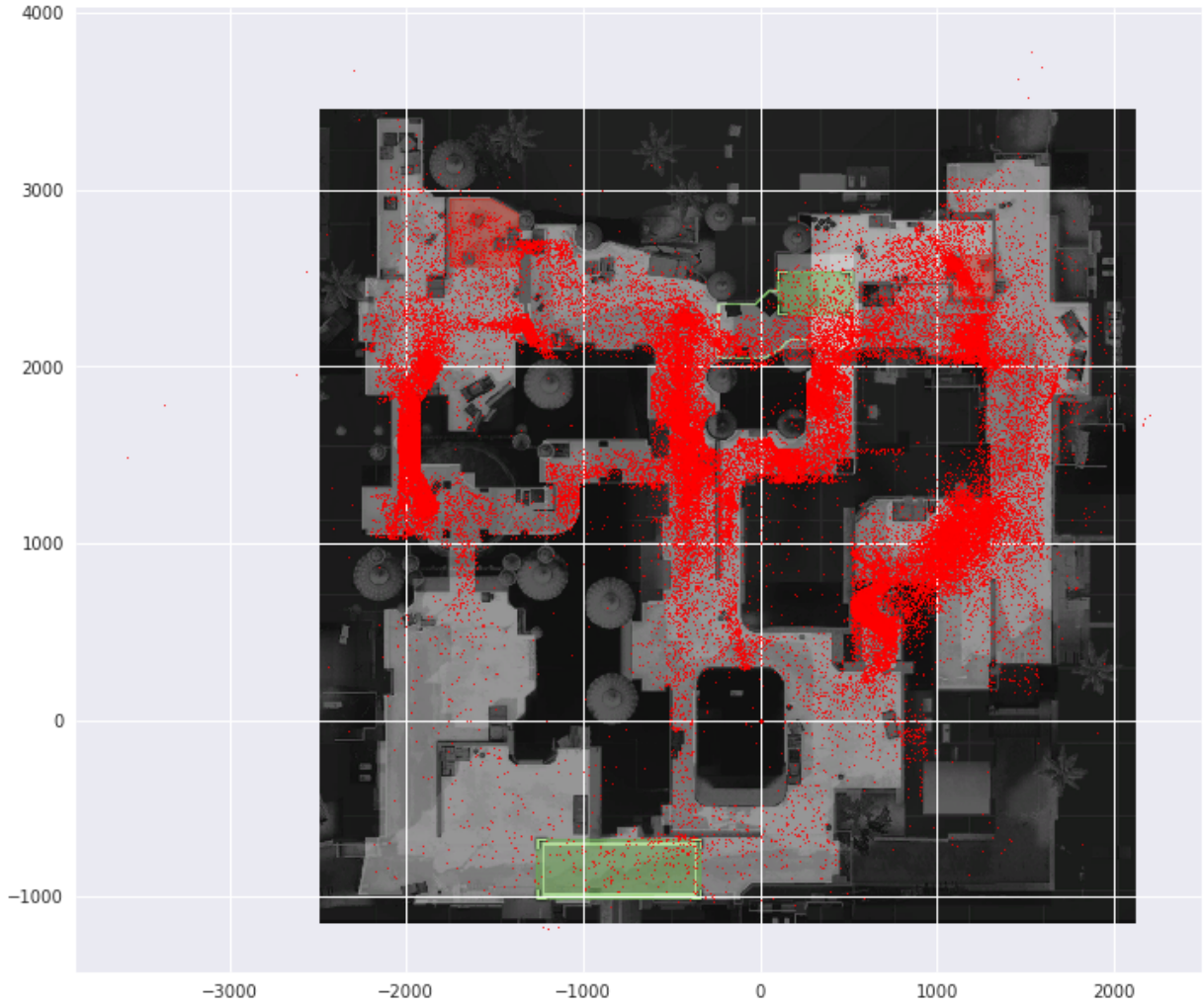
Out[21]: <matplotlib.collections.PathCollection at 0x7f949d8d12b0>



Grenade plot where there was no victim

In [22]:
```
#use victim instead of hp_dmg because victim includes flashes while hp_dmg and ar
noVictim = dust_data.loc[dust_data['victim'] == False]

#plot damaging grenades
img = plt.imread("de_dust2.png")
fig, ax = plt.subplots()
ax.imshow(img)
fig.set_size_inches(18.5, 10.5)
#dust size meta data: 2127  3455    1024    1024    -2486   -1150
ax.imshow(img, extent=[-2486,2127,-1150,3455])
plt.scatter(noVictim['nade_land_x'], noVictim['nade_land_y'], s=0.5, c ='r')
```

Out[22]: <matplotlib.collections.PathCollection at 0x7f949d906b38>



Machine learning using CSGO dataset

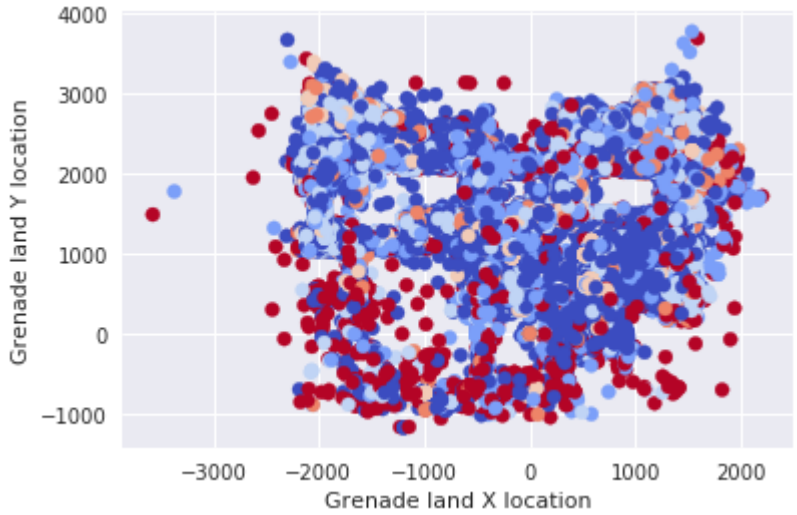
using flower data set for example of Support Vector Machines.

In [23]:
```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn import svm, datasets
import matplotlib.pyplot as plt
```

In [24]:
```
from sklearn import datasets
```

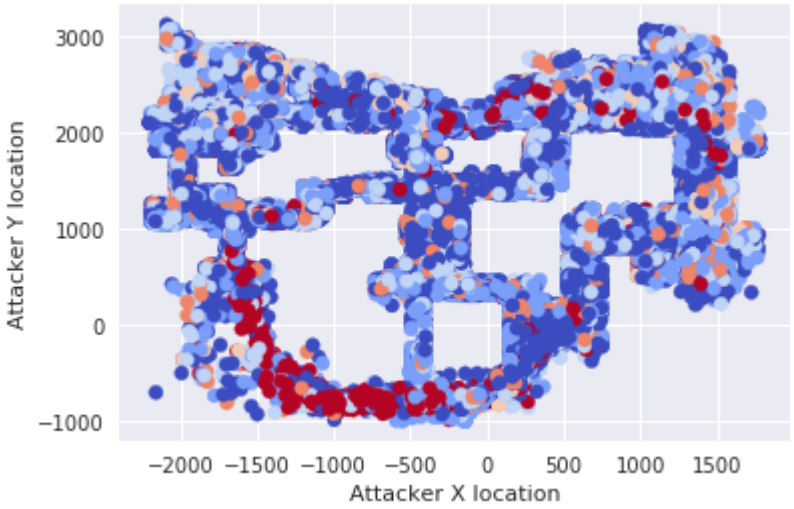
In [25]:
```
X = dust_data['nade_land_x']
Y = dust_data['nade_land_y']
C = dust_data['nade_type']

plt.scatter(X, Y, c=C,cmap= plt.cm.coolwarm)
plt.xlabel('Grenade land X location')
plt.ylabel('Grenade land Y location')
```

Out[25]: Text(0,0.5,'Grenade land Y location')



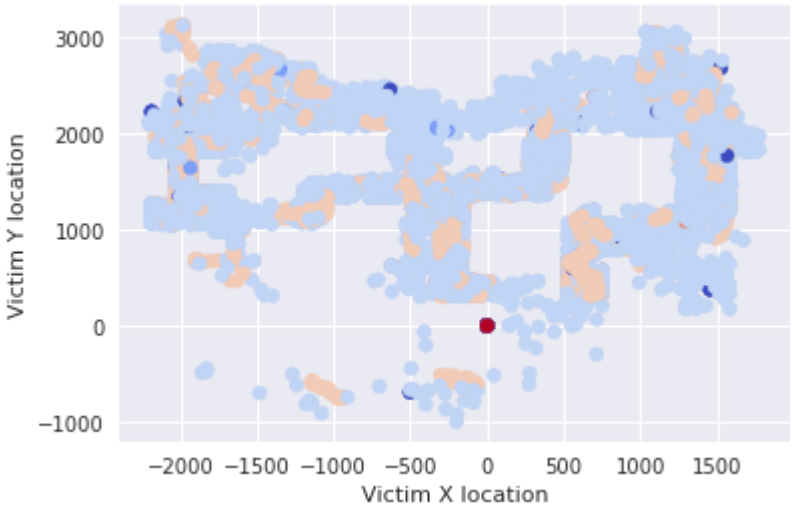
In [26]:
```
X = dust_data['att_pos_x']
Y = dust_data['att_pos_y']
C = dust_data['nade_type']

plt.scatter(X, Y, c=C,cmap= plt.cm.coolwarm)
plt.xlabel('Attacker X location')
plt.ylabel('Attacker Y location')
```

Out[26]: Text(0,0.5,'Attacker Y location')



In [27]:
```
X = dust_data['vic_pos_x']
Y = dust_data['vic_pos_y']
C = dust_data['nade_type']

plt.scatter(X, Y, c=C,cmap= plt.cm.coolwarm)
plt.xlabel('Victim X location')
plt.ylabel('Victim Y location')
```

Out[27]: Text(0,0.5,'Victim Y location')



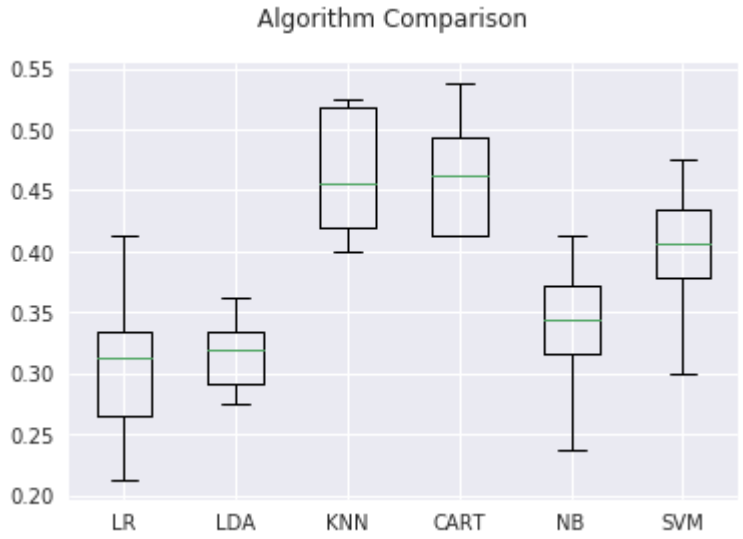
Decide which machine learning model to use

Attacker position

In [28]:
```
# Spot Check Algorithms
seed = 7
scoring = 'accuracy'
validation_size = 0.20

X = dust_data[:1000].iloc[:,24:26]
Y = dust_data[:1000].iloc[:,35]

X, X_validation, Y, Y_validation = model_selection.train_test_split(X, Y, test_si

models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=s
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.306250 (0.055410)
LDA: 0.315000 (0.050559)
KNN: 0.465000 (0.048348)
CART: 0.447500 (0.067500)
NB: 0.336250 (0.051675)
SVM: 0.398750 (0.054900)
```

In [29]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



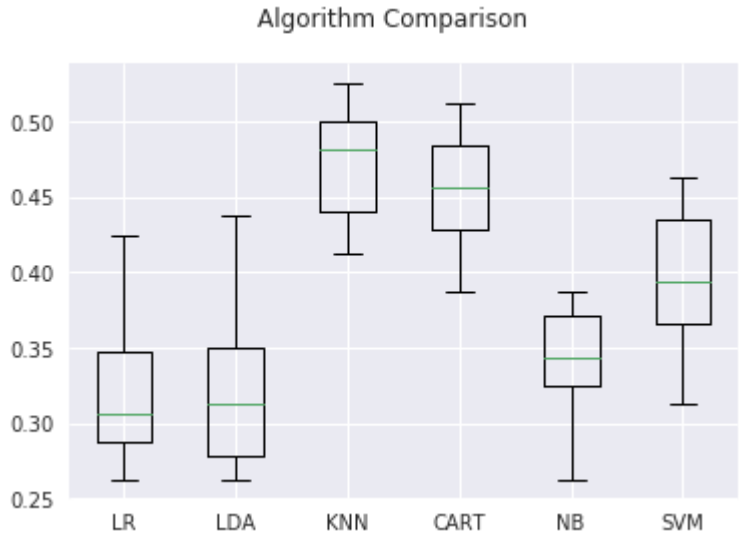
Best machine learning algorithm for attacker position: K Neighbors Classification

Grenade landing position

In [30]:
```
# Spot Check Algorithms
seed = 7
scoring = 'accuracy'
validation_size = 0.20

X = dust_data[:1000].iloc[:,26:28]
Y = dust_data[:1000].iloc[:,35]

X, X_validation, Y, Y_validation = model_selection.train_test_split(X, Y, test_si

models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=s
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.320000 (0.044791)
LDA: 0.321250 (0.050327)
KNN: 0.471250 (0.038750)
CART: 0.452500 (0.039051)
NB: 0.350000 (0.050312)
SVM: 0.392500 (0.050990)
```

In [31]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



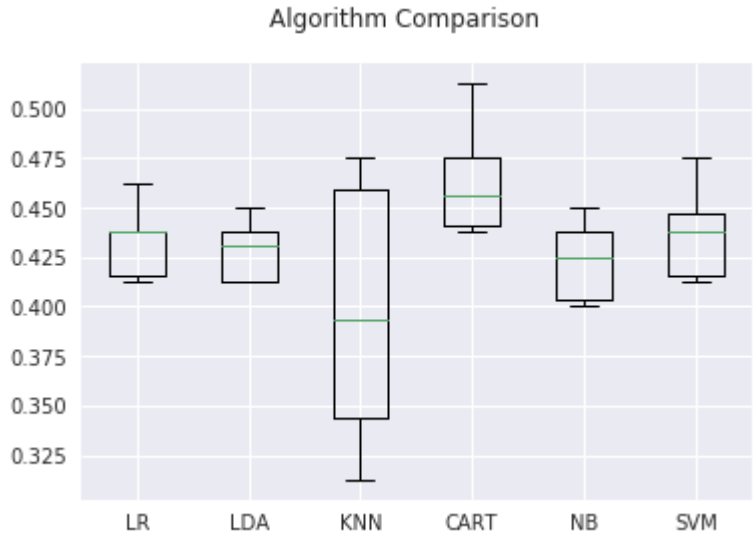
Best machine learning algorithm for grenade landing position: K Neighbors Classification

Victim position

In [32]:
```
# Spot Check Algorithms
seed = 7
scoring = 'accuracy'
validation_size = 0.20

X = dust_data[:1000].iloc[:,28:30]
Y = dust_data[:1000].iloc[:,35]

X, X_validation, Y, Y_validation = model_selection.train_test_split(X, Y, test_si

models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=s
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.417500 (0.041908)
LDA: 0.412500 (0.039131)
KNN: 0.397500 (0.058843)
CART: 0.442500 (0.059739)
NB: 0.410000 (0.042131)
SVM: 0.421250 (0.041852)
```

In [33]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Best machine learning algorithm for attacker position: DecisionTreeClassifier (CART)

Predictions using each Machine Learning Algorithm

Define variables for use with each algorithm

In [34]:
```
att = dust_data[:1000].iloc[:,24:26]
land = dust_data[:1000].iloc[:,26:28]
vic = dust_data[:1000].iloc[:,28:30]

type = dust_data[:1000].iloc[:,35]
```

Logistic Regression

In [35]:
```
# Make predictions on validation dataset
r = LogisticRegression()
r.fit(att, type)
predictions = r.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.4
[[71  0  0  0  0  0]
 [34  0  0  0  0  0]
 [43  0  9  0  0  0]
 [11  0  1  0  0  0]
 [19  0  0  0  0  0]
 [12  0  0  0  0  0]]
             precision    recall  f1-score   support

          0       0.37      1.00      0.54        71
          1       0.00      0.00      0.00        34
          2       0.90      0.17      0.29        52
          3       0.00      0.00      0.00        12
          4       0.00      0.00      0.00        19
          5       0.00      0.00      0.00        12

avg / total       0.37      0.40      0.27       200
```

LinearDiscriminantAnalysis

In [36]:
```
# Make predictions on validation dataset
lda = LinearDiscriminantAnalysis()
lda.fit(att, type)
predictions = lda.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.405
[[71  0  0  0  0  0]
 [34  0  0  0  0  0]
 [42  0 10  0  0  0]
 [11  0  1  0  0  0]
 [19  0  0  0  0  0]
 [12  0  0  0  0  0]]
             precision    recall  f1-score   support

          0       0.38      1.00      0.55        71
          1       0.00      0.00      0.00        34
          2       0.91      0.19      0.32        52
          3       0.00      0.00      0.00        12
          4       0.00      0.00      0.00        19
          5       0.00      0.00      0.00        12

avg / total       0.37      0.41      0.28       200
```

KNN (K Neighbors Classifier)

In [37]:
```
# Make predictions on validation dataset
knn = KNeighborsClassifier()
knn.fit(att, type)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.195
[[ 0 71  0  0  0  0]
 [ 0 34  0  0  0  0]
 [12 33  5  1  1  0]
 [ 5  7  0  0  0  0]
 [ 0 19  0  0  0  0]
 [ 0 12  0  0  0  0]]
             precision    recall  f1-score   support

          0       0.00      0.00      0.00        71
          1       0.19      1.00      0.32        34
          2       1.00      0.10      0.18        52
          3       0.00      0.00      0.00        12
          4       0.00      0.00      0.00        19
          5       0.00      0.00      0.00        12

avg / total       0.29      0.20      0.10       200
```

DecisionTreeClassifier (CART)

In [38]:
```
# Make predictions on validation dataset
dtc = DecisionTreeClassifier()
dtc.fit(att, type)
predictions = dtc.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.2
[[ 0 71  0  0  0  0]
 [ 0 34  0  0  0  0]
 [ 8 34  6  0  2  2]
 [ 2 10  0  0  0  0]
 [ 0 19  0  0  0  0]
 [ 0 12  0  0  0  0]]
             precision    recall  f1-score   support

          0       0.00      0.00      0.00        71
          1       0.19      1.00      0.32        34
          2       1.00      0.12      0.21        52
          3       0.00      0.00      0.00        12
          4       0.00      0.00      0.00        19
          5       0.00      0.00      0.00        12

avg / total       0.29      0.20      0.11       200
```

GaussianNB

In [39]:
```
# Make predictions on validation dataset
gnb = GaussianNB()
gnb.fit(att, type)
predictions = gnb.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.415
[[71  0  0  0  0  0]
 [34  0  0  0  0  0]
 [40  0 12  0  0  0]
 [11  0  1  0  0  0]
 [19  0  0  0  0  0]
 [12  0  0  0  0  0]]
             precision    recall  f1-score   support

          0       0.38      1.00      0.55        71
          1       0.00      0.00      0.00        34
          2       0.92      0.23      0.37        52
          3       0.00      0.00      0.00        12
          4       0.00      0.00      0.00        19
          5       0.00      0.00      0.00        12

avg / total       0.37      0.41      0.29       200
```

Support Vector Classification

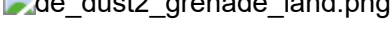
In [40]:
```
# Make predictions on validation dataset
svc = SVC()
svc.fit(att, type)
predictions = svc.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.355
[[71  0  0  0  0  0]
 [34  0  0  0  0  0]
 [52  0  0  0  0  0]
 [12  0  0  0  0  0]
 [19  0  0  0  0  0]
 [12  0  0  0  0  0]]
             precision    recall  f1-score   support

          0       0.35      1.00      0.52        71
          1       0.00      0.00      0.00        34
          2       0.00      0.00      0.00        52
          3       0.00      0.00      0.00        12
          4       0.00      0.00      0.00        19
          5       0.00      0.00      0.00        12

avg / total       0.13      0.35      0.19       200
```

Predictions for attacking location

In [41]:
```
#Initialize all algorithms
r = LogisticRegression().fit(att, type)
lda = LinearDiscriminantAnalysis().fit(att, type)
knn = KNeighborsClassifier().fit(att, type)
dtc = DecisionTreeClassifier().fit(att, type)
gnb = GaussianNB().fit(att, type)
svm = SVC(kernel='rbf', C=0.5, probability=True).fit(att, type)
```

Double A Doors (X: 600, Y:400)


de_dust2_grenade_land.png

In [61]:
```
# Predict probability
print('Predict grenades thrown from Double A Doors (near 600,400)')
print()
print(r.predict_proba([[600,400]]))
print(lda.predict_proba([[600,400]]))
print(knn.predict_proba([[600,400]]))
print(dtc.predict_proba([[600,400]]))
print(gnb.predict_proba([[600,400]]))
print(svm.predict_proba([[600,400]]))
```

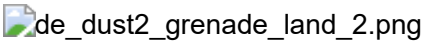
```
Predict grenades thrown from Double A Doors (near 600,400)

[[0.04915996 0.04150215 0.74657585 0.13380482 0.01578406 0.01317316]]
[[0.27155556 0.20845896 0.31254937 0.09175864 0.06358015 0.05209731]]
[[0.  0.  0.4 0.6 0.  0. ]]
[[0. 0. 1. 0. 0. 0.]]
[[0.         0.         0.72422502 0.27577498 0.         0.        ]]
[[0.01941364 0.01892298 0.6138074  0.2658237  0.0443227  0.03770957]]
```

In [43]:
```
# Predict number
print(r.predict([[600,400]]))
print(lda.predict([[600,400]]))
print(knn.predict([[600,400]]))
print(dtc.predict([[600,400]]))
print(gnb.predict([[600,400]]))
print(svm.predict([[600,400]]))
```

```
[0]
[0]
[0]
[0]
[0]
[0]
```

Important note: Because the data in the dataset is the exact location (ie: 1812.132934, 843.2344556) it will not predict any number of grenades thrown from a certain location. (unless we get extremely lucky which you will see in some predictions)

B Upper Hallway (X: 1800, Y:1100)



In [44]:
```
# Predict probability
print('Predict grenades thrown from B Upper Hallway (near -1800,1100)')
print()
print(r.predict_proba([[1800,1100]]))
print(lda.predict_proba([[1800,1100]]))
print(knn.predict_proba([[1800,1100]]))
print(dtc.predict_proba([[1800,1100]]))
print(gnb.predict_proba([[1800,1100]]))
print(svm.predict_proba([[1800,1100]]))
```

```
Predict grenades thrown from B Upper Hallway (near -1800,1100)

[[0.30358464 0.23322122 0.27225017 0.07783783 0.07689079 0.03621534]]
[[0.2955916  0.23862344 0.26402626 0.07903203 0.07966124 0.04306542]]
[[0.  0.8 0.2 0.  0.  0. ]]
[[0. 0. 1. 0. 0. 0.]]
[[0.25644236 0.22350099 0.34149534 0.08930284 0.07356429 0.01569418]]
[[0.34934426 0.24148224 0.24774017 0.03758558 0.0694424  0.05440535]]
```

In [45]:
```
# Predict number thrown
print('Predict grenades thrown from B Upper Hallway (near -1800,1100)')
print()
print(r.predict([[1800,1100]]))
print(lda.predict([[1800,1100]]))
print(knn.predict([[1800,1100]]))
print(dtc.predict([[1800,1100]]))
print(gnb.predict([[1800,1100]]))
print(svm.predict([[1800,1100]]))
```

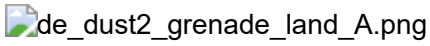
```
Predict grenades thrown from B Upper Hallway (near -1800,1100)

[0]
[0]
[1]
[2]
[2]
[0]
```

Important note: Because the data in the dataset is the exact location (ie: 1812.132934, 843.2344556) it will not predict any number of grenades thrown from a certain location. (unless we get extremely lucky which you will see in some predictions)

Predictions for grenade landing location

In [46]:
```
#Initialize all algorithms
r = LogisticRegression().fit(land, type)
lda = LinearDiscriminantAnalysis().fit(land, type)
knn = KNeighborsClassifier().fit(land, type)
dtc = DecisionTreeClassifier().fit(land, type)
gnb = GaussianNB().fit(land, type)
svm = SVC(kernel='rbf', C=0.5, probability=True).fit(land, type)
```

Grenades landed on Bombsite A



In [47]:
```
# Predict probability
print('Predict grenades landing on bombsite A (near 1050,2300)')
print()
print(r.predict_proba([[1050,2300]]))
print(lda.predict_proba([[1050,2300]]))
print(knn.predict_proba([[1050,2300]]))
print(dtc.predict_proba([[1050,2300]]))
print(gnb.predict_proba([[1050,2300]]))
print(svm.predict_proba([[1050,2300]]))
```

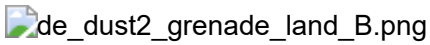
```
Predict grenades landing on bombsite A (near 1050,2300)

[[0.28258485 0.27920005 0.26943641 0.02409189 0.12669493 0.01799186]]
[[0.27144874 0.26626689 0.25942603 0.03045634 0.14862841 0.02377359]]
[[0.4 0.2 0.4 0.  0.  0. ]]
[[1. 0. 0. 0. 0. 0.]]
[[0.27704543 0.27628763 0.26745564 0.02736682 0.12529121 0.02655327]]
[[0.35003951 0.25100046 0.24319131 0.03684609 0.06434059 0.05458205]]
```

In [48]:
```
# Predict number thrown
print('Predict grenades landing on bombsite A (near 1050,2300)')
print()
print(r.predict([[1050,2300]]))
print(lda.predict([[1050,2300]]))
print(knn.predict([[1050,2300]]))
print(dtc.predict([[1050,2300]]))
print(gnb.predict([[1050,2300]]))
print(svm.predict([[1050,2300]]))
```

```
Predict grenades landing on bombsite A (near 1050,2300)

[0]
[0]
[0]
[0]
[0]
[0]
```

Important note: Because the data in the dataset is the exact location (ie: 1812.132934, 843.2344556) it will not predict any number of grenades thrown from a certain location. (unless we get extremely lucky which you will see in some predictions)

Grenades laned on Bombsite B



In [49]:
```
# Predict probability
print('Predict grenades landing on bombsite B (near -1500,2500)')
print()
print(r.predict_proba([[-1500,2500]]))
print(lda.predict_proba([[-1500,2500]]))
print(knn.predict_proba([[-1500,2500]]))
print(dtc.predict_proba([[-1500,2500]]))
print(gnb.predict_proba([[-1500,2500]]))
print(svm.predict_proba([[-1500,2500]]))
```

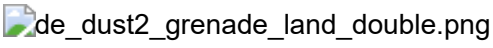
```
Predict grenades landing on bombsite B (near -1500,2500)

[[0.23179268 0.33094773 0.28738134 0.04851617 0.06479121 0.03657086]]
[[0.23989082 0.30430951 0.27423597 0.05712926 0.07711776 0.04731669]]
[[0.8 0.  0.2 0.  0.  0. ]]
[[0. 0. 1. 0. 0. 0.]]
[[0.22225483 0.30442542 0.30829917 0.04217585 0.05796073 0.064884  ]]
[[0.35003951 0.25100046 0.24319131 0.03684609 0.06434059 0.05458205]]
```

In [50]:
```
# Predict probability
print('Predict grenades landing on bombsite B (near -1500,2500)')
print()
print(r.predict([[-1500,2500]]))
print(lda.predict([[-1500,2500]]))
print(knn.predict([[-1500,2500]]))
print(dtc.predict([[-1500,2500]]))
print(gnb.predict([[-1500,2500]]))
print(svm.predict([[-1500,2500]]))
```

```
Predict grenades landing on bombsite B (near -1500,2500)

[1]
[1]
[0]
[2]
[2]
[0]
```

Important note: Because the data in the dataset is the exact location (ie: 1812.132934, 843.2344556) it will not predict any number of grenades thrown from a certain location. (unless we get extremely lucky which you will see in some predictions)

Grenades laned on Long Double Doors



In [51]:
```
# Predict probability
print('Predict grenades landing on Long Double doors (near -400,1600)')
print()
print(r.predict_proba([[-400,1600]]))
print(lda.predict_proba([[-400,1600]]))
print(knn.predict_proba([[-400,1600]]))
print(dtc.predict_proba([[-400,1600]]))
print(gnb.predict_proba([[-400,1600]]))
print(svm.predict_proba([[-400,1600]]))
```

```
Predict grenades landing on Long Double doors (near -400,1600)

[[0.32188318 0.24158924 0.26118536 0.06701056 0.05780509 0.05052657]]
[[0.30125562 0.2422264  0.25973221 0.07690259 0.06117519 0.05870799]]
[[0.2 0.2 0.4 0.2 0.  0. ]]
[[0. 0. 0. 0. 0. 1.]]
[[0.32053518 0.29196991 0.2398225  0.06162402 0.05042646 0.03562192]]
[[0.35003951 0.25100046 0.24319131 0.03684609 0.06434059 0.05458205]]
```

In [52]:
```
# Predict probability
print('Predict grenades landing on Long Double doors (near -400,1600)')
print()
print(r.predict([[-400,1600]]))
print(lda.predict([[-400,1600]]))
print(knn.predict([[-400,1600]]))
print(dtc.predict([[-400,1600]]))
print(gnb.predict([[-400,1600]]))
print(svm.predict([[-400,1600]]))
```

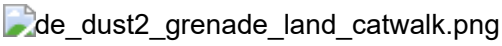
```
Predict grenades landing on Long Double doors (near -400,1600)

[0]
[0]
[2]
[5]
[0]
[0]
```

Important note: Because the data in the dataset is the exact location (ie: 1812.132934, 843.2344556) it will not predict any number of grenades thrown from a certain location. (unless we get extremely lucky which you will see in some predictions)

Prediction for Victim locations

In [53]:
```
#Initialize all algorithms
r = LogisticRegression().fit(vic, type)
lda = LinearDiscriminantAnalysis().fit(vic, type)
knn = KNeighborsClassifier().fit(vic, type)
dtc = DecisionTreeClassifier().fit(vic, type)
gnb = GaussianNB().fit(vic, type)
svm = SVC(kernel='rbf', C=0.5, probability=True).fit(vic, type)
```

Victims on A Catwalk



In [54]:
```
# Predict probability
print(r.predict_proba([[400,2000]]))
print(lda.predict_proba([[400,2000]]))
print(knn.predict_proba([[400,2000]]))
print(dtc.predict_proba([[400,2000]]))
print(gnb.predict_proba([[400,2000]]))
print(svm.predict_proba([[400,2000]]))
```

```
[[8.90641173e-06 1.67758300e-05 8.46324899e-01 1.53600619e-01
  2.54837131e-05 2.33165134e-05]]
[[0.04471382 0.03305325 0.6603498  0.24502581 0.00929117 0.00756615]]
[[0. 0. 1. 0. 0. 0.]]
[[0. 0. 1. 0. 0. 0.]]
[[0.         0.         0.94382822 0.05617178 0.         0.        ]]
[[0.01941364 0.01892298 0.6138074  0.2658237  0.0443227  0.03770957]]
```

In [55]:
```
# Predict number of victims
print(r.predict([[400,2000]]))
print(lda.predict([[400,2000]]))
print(knn.predict([[400,2000]]))
print(dtc.predict([[400,2000]]))
print(gnb.predict([[400,2000]]))
print(svm.predict([[400,2000]]))
```

```
[2]
[2]
[2]
[2]
[2]
[2]
[2]
```

Important note: Because the data in the dataset is the exact location (ie: 1812.132934, 843.2344556) it will not predict any number of grenades thrown from a certain location. (unless we get extremely lucky which you will see in some predictions)

Random Code snippets

In [56]:
```
#dust_ten = dust_ten.fillna(0)
dust_data.isna().apply(lambda x: sum(x))[:5]
```

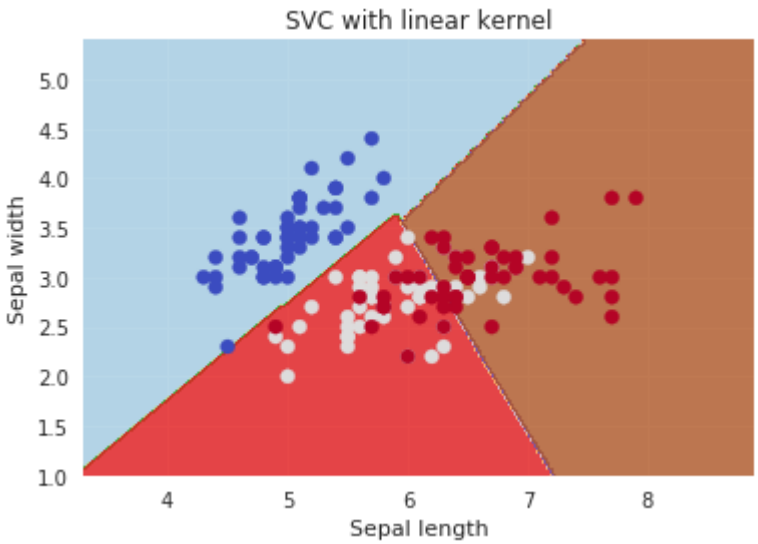
Out[56]:
```
Unnamed: 0      0
Unnamed: 0.1    0
file            0
map             0
round           0
dtype: int64
```

In [57]:
```
dust_ten = dust_data[:1000]
dust_ten.shape
```

Out[57]: (1000, 36)

In [58]:
```
#we cannot really tell which type of grenade is going to be thrown in what positi
#so we will use SVM machine learning method to classify each grenade and use
#this svm to predict where each type of grenade would likely be thrown in a round
#SMV will help us make sense of that condensed scatter plo
```

In [59]:
```
# import some data to play with
h = 0.2
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
C = 1.0 # SVM regularization parameter
svc = SVC(kernel='linear', C=1).fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```




```
#SVM graphing.. too large and slow for codio/laptop

from sklearn.preprocessing import MinMaxScaler
```

```
scaling = MinMaxScaler(feature_range=(-1000,1000)).fit(X)
X = scaling.transform(X)
X

h = 0.2
C = 1.0 # SVM regularization parameter

#X.reshape(1, -1)

svc = svm.SVC(kernel='rbf', C=1).fit(X, Y)
#create a mesh to plot in
x_min, x_max = -2486, 2127
y_min, y_max = -1150, 3455
h = (x_max / x_min)/100

print(x_min, x_max, y_min, y_max, h)
xx, yy = np.meshgrid(np.arange(x_min, x_max),
 np.arange(y_min, y_max))
plt.subplot(1, 1, 1)

print(xx.shape)
print(yy.shape)
print(X.shape)
print(Y.shape)

Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X['att_pos_x'], X['att_pos_y'], c=Y, cmap=plt.cm.coolwarm)
plt.xlabel('X location')
plt.ylabel('Y location')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with rbf kernel')
plt.show()
```

In [ ]: