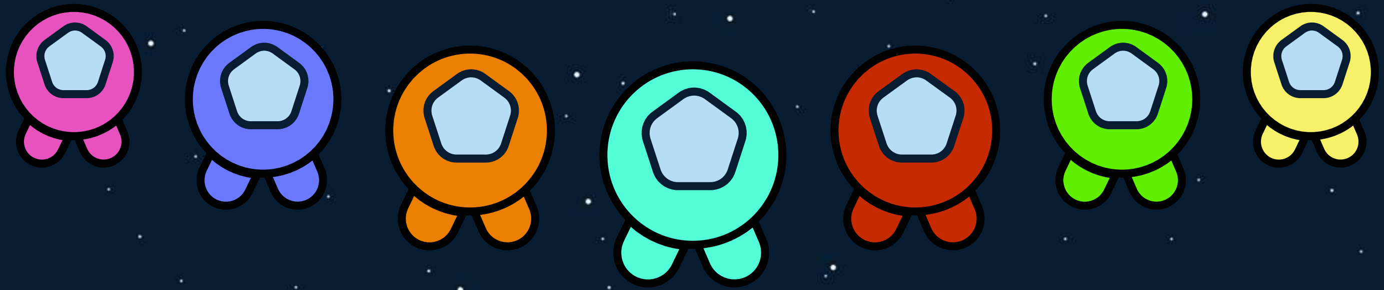
Real Time and Adaptive Gamer Type Classification on Space War

ChemEng



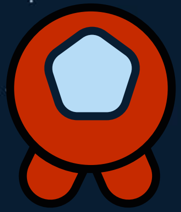
Chem Eng Members

Pongsarat C.

6310412018

Nidchapan N.

6310412022



Saranchai A.

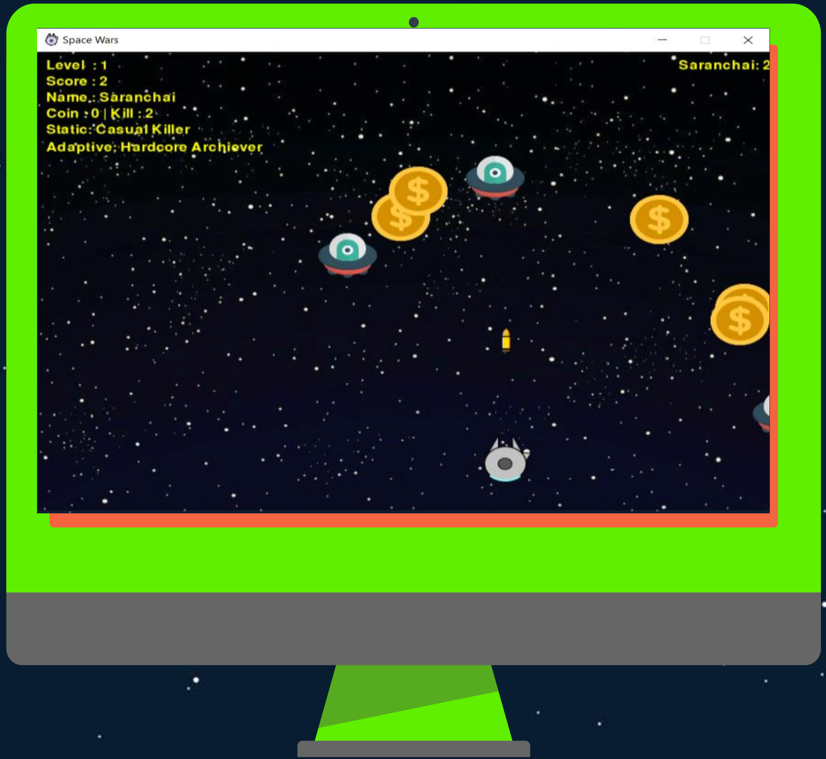
6310412024

Contents

- Game: Space War
- Project Objective
- Model in detail
- Suggestion

Game: Space War

Space war game is a spacecraft shooting game with an objective to kill enemy or collect the coin as much as possible to gain the highest score



Space Wars

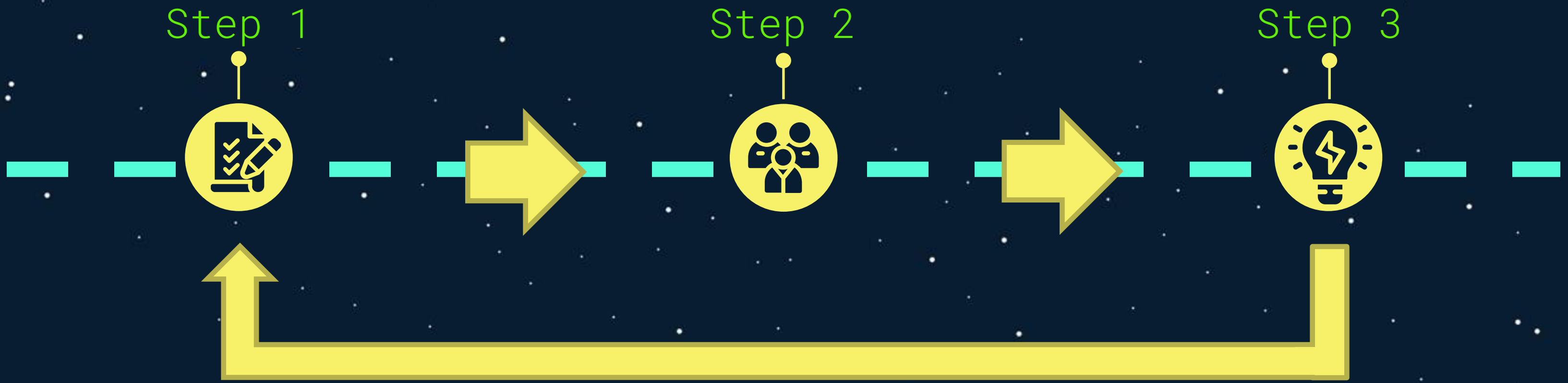
Level : 1
Score : 2
Name.: Saranchai
Coin : 0 | Kill : 2
Static: Casual Killer
Adaptive: Hardcore Archiever

Saranchai: 2

Project Objective

- To make a real time and adaptive classification model that instantaneously classify the player to the most suit gamer type based on their playing style in real time
- As the game progresses, the model will
  - make a gamer type prediction
  - learn by itself from fresh player data
- Gamer type divided into 4 classes: “Hardcore Killer”, “Hardcore Achiever”, “Casual Killer” and “Casual Achiever”

Model Overview



Prepare the game feature

Step 1

Let model predict the player's cluster and gamer type

Step 2

Let model learn new the feature

Step 3

After every 240 game loop passed,
the model will make a prediction and learning by itself again

Model Selection

Model	Reason
CluStream	Less adaptive comparing to Incremental Kmeans from trial
Incremental Kmeans	More adaptive comparing to CluStream from trial ✓
StreamKMeans	Incompatible with Space War game (error)

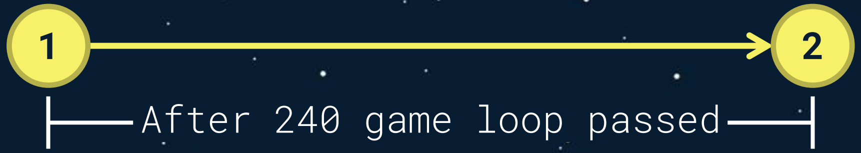
Available Features

Variable	Description
A0	Average position in X axis
A1	Average position in Y axis
A2	Total number of coins collected
A3	Total number of destroyed enemies
A4	Total number of shots
A5	Total number of shots without enemies (A4 - A3)
A6	Level reach
A7	Key X pressed count
A8	Key Y pressed count
A9*	Number of enemy created
A10*	Number of coin created

* The value of the feature is not align with the actual game play

Feature Selection

Variable	Description	Calculation
Amount of coin increase	Number of enemy kill increase within 240 game loop	**(Total number of coins collected$_2$ - Total number of coins collected$_1$)**
Amount of kill increase	Number of coin increase within 240 game loop	**(Total number of destroyed enemies$_2$ -Total number of destroyed enemies$_1$)**



1

2

After 240 game loop passed

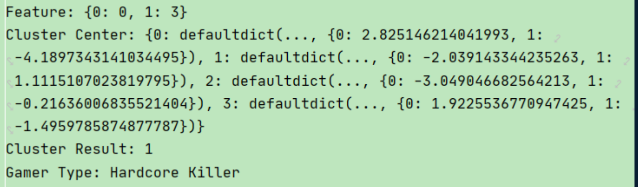
- Number 240 tends to give a best model adaptivity compare to 120 and 360 (judged by team)

Player Classification

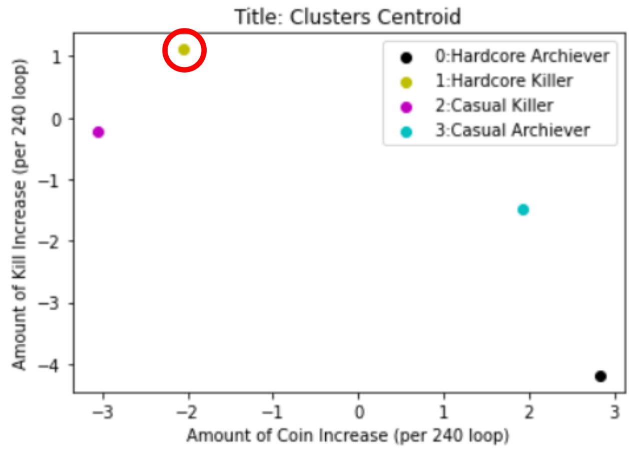
- **Amount of coin increase** and **Amount of kill increase** are used as a model feature

- The centroid of each cluster will be represented as Amount of Coin increase(x) and Amount of kill increase(y) coordinate

- The cluster with highest y-centroid will be assigned to “Hardcore Killer”



```
Feature: {0: 0, 1: 3}
Cluster Center: {0: defaultdict(..., {0: 2.825146214041993, 1:
 -4.189734314103449 5}), 1: defaultdict(..., {0: -2.039143344235263, 1:
 1.1115107023819795}), 2: defaultdict(..., {0: -3.049046682564213, 1:
 -0.21636006835521404}), 3: defaultdict(..., {0: 1.9225536770947425, 1:
 -1.4959785874877787})}
Cluster Result: 1
Gamer Type: Hardcore Killer
```

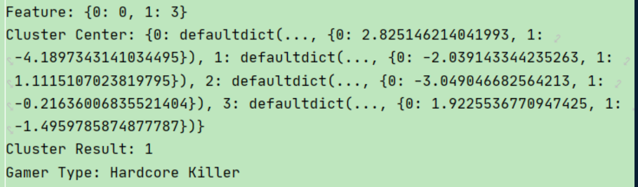


Player Classification

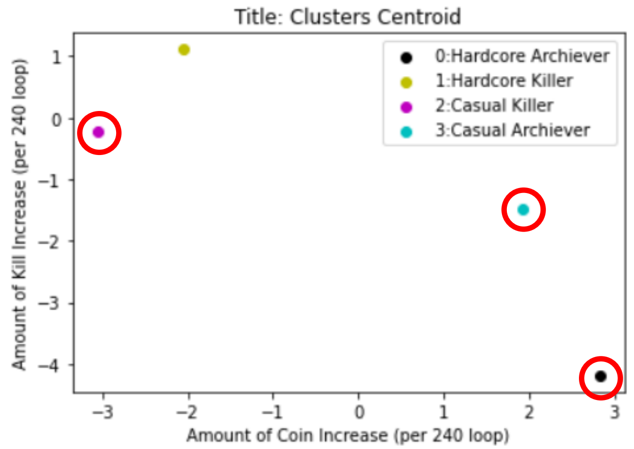
- In a similar way, the cluster with highest x-centroid will be assigned to "Hardcore Achiever".

  (If the highest x-centroid cluster is the same one as highest y-centroid the second highest x-centroid will be assign instead)

- The cluster which has a higher x-centroid from the last two will be assigned to "Casual Killer" and the last cluster will be "Casual Achiever"



```
Feature: {0: 0, 1: 3}
Cluster Center: {0: defaultdict(..., {0: 2.825146214041993, 1:
 -4.189734314103449 5}), 1: defaultdict(..., {0: -2.039143344235263, 1:
 1.1115107023819795}), 2: defaultdict(..., {0: -3.049046682564213, 1:
 -0.21636006835521404}), 3: defaultdict(..., {0: 1.9225536770947425, 1:
 -1.4959785874877787})}
Cluster Result: 1
Gamer Type: Hardcore Killer
```

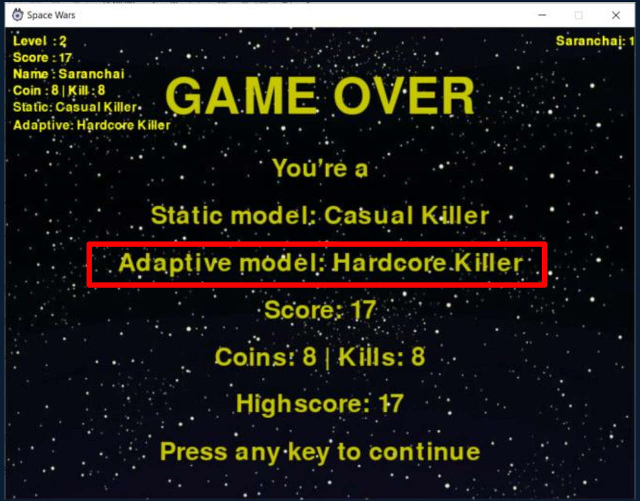


Game Over Player Classification

- Every time the model predicts the player type, the data will be recorded

- Final player type is the most assigned player type

- After the game over, the result of final player type will be shown on the game over screen



Level : 2
Score : 17
Name : Saranchai
Coin : 8 | Kill : 8
Static: Casual Killer
Adaptive: Hardcore Killer

Saranchai: 1

GAME OVER

You're a

Static model: Casual Killer

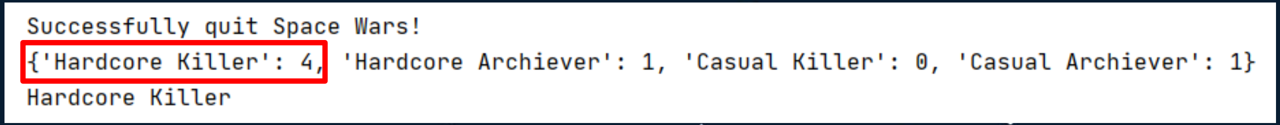
Adaptive model: Hardcore Killer

Score: 17

Coins: 8 | Kills: 8

Highscore: 17

Press any key to continue



```
Successfully quit Space Wars!
{'Hardcore Killer': 4, 'Hardcore Archiever': 1, 'Casual Killer': 0, 'Casual Archiever': 1}
Hardcore Killer
```

Results

Player	Quest.	Trial 1		Trial 2		Trial 3	
		Static	Adaptive	Static	Adaptive	Static	Adaptive
Pongsarat	HA	HK	HA	HA	HA	HA	HA
Nidchapan	CA	HK	HK	HK	HK	HK	HK
Saranchai	CA	HA	HK	HA	HK	HA	HK

Model	Accuracy
Static	22.22%
Adaptive	33.33%

HK = Hardcore Killer
HA = Hardcore Achiever
CK = Casual Killer
CA = Casual Achiever

Suggestion

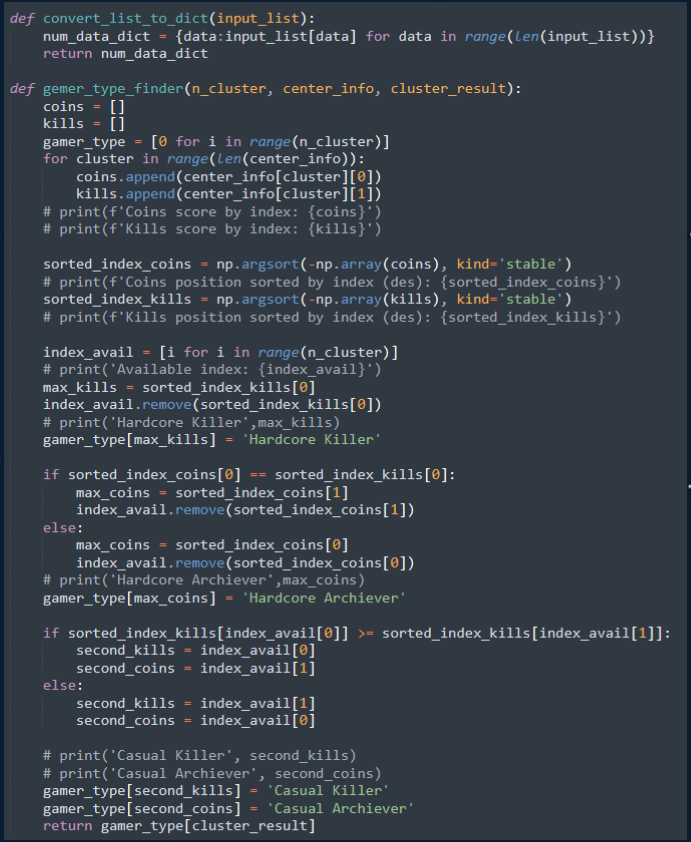
1. Use better model feature such as %coin collected and %enemy killed in a period of time

   - **%coin collected = #coin collected / #coin respawn** (in a period of time)

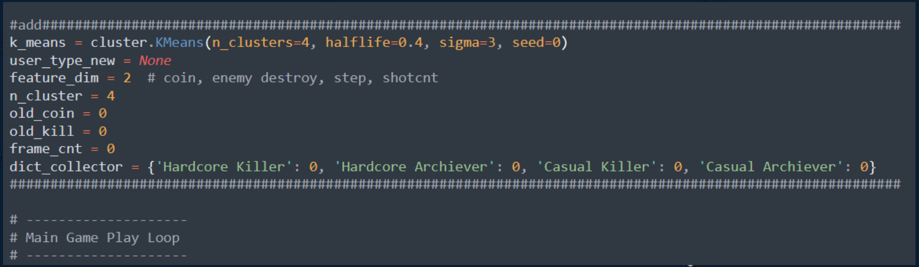
   - **%enemy killed = #enemy killed / #enemy respawn** (in a period of time)

2. Notice that, as a game level increase, it is hard to survive the game without killing a lot of enemy and this could lead the model to assign the most players as "Hardcore Killer"

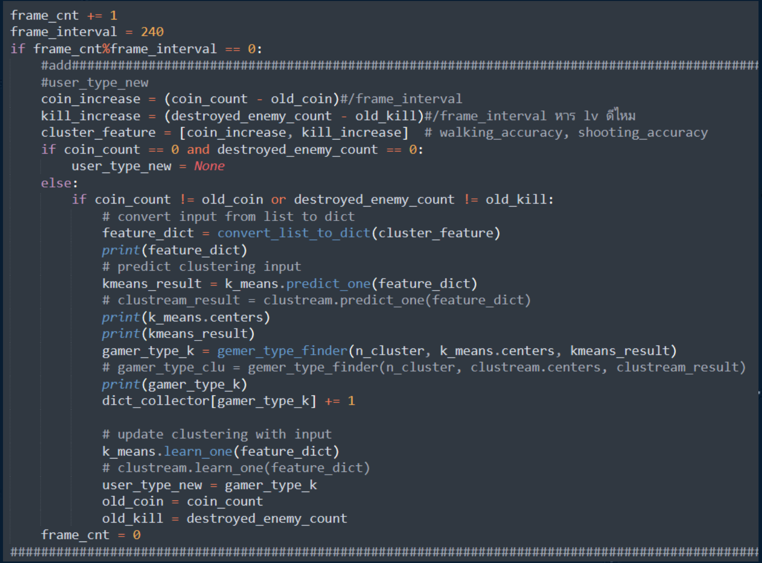
Code



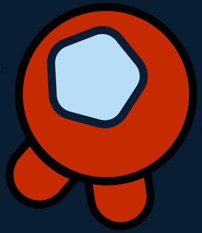
```python
def convert_list_to_dict(input_list):
    num_data_dict = {data:input_list[data] for data in range(len(input_list))}
    return num_data_dict

def gemer_type_finder(n_cluster, center_info, cluster_result):
    coins = []
    kills = []
    gamer_type = [0 for i in range(n_cluster)]
    for cluster in range(len(center_info)):
        coins.append(center_info[cluster][0])
        kills.append(center_info[cluster][1])
    # print(f'Coins score by index: {coins}')
    # print(f'Kills score by index: {kills}')

    sorted_index_coins = np.argsort(-np.array(coins), kind='stable')
    # print(f'Coins position sorted by index (des): {sorted_index_coins}')
    sorted_index_kills = np.argsort(-np.array(kills), kind='stable')
    # print(f'Kills position sorted by index (des): {sorted_index_kills}')

    index_avail = [i for i in range(n_cluster)]
    # print('Available index: {index_avail}')
    max_kills = sorted_index_kills[0]
    index_avail.remove(sorted_index_kills[0])
    # print('Hardcore Killer',max_kills)
    gamer_type[max_kills] = 'Hardcore Killer'

    if sorted_index_coins[0] == sorted_index_kills[0]:
        max_coins = sorted_index_coins[1]
        index_avail.remove(sorted_index_coins[1])
    else:
        max_coins = sorted_index_coins[0]
        index_avail.remove(sorted_index_coins[0])
    # print('Hardcore Archiever',max_coins)
    gamer_type[max_coins] = 'Hardcore Archiever'

    if sorted_index_kills[index_avail[0]] >= sorted_index_kills[index_avail[1]]:
        second_kills = index_avail[0]
        second_coins = index_avail[1]
    else:
        second_kills = index_avail[1]
        second_coins = index_avail[0]

    # print('Casual Killer', second_kills)
    # print('Casual Archiever', second_coins)
    gamer_type[second_kills] = 'Casual Killer'
    gamer_type[second_coins] = 'Casual Archiever'
    return gamer_type[cluster_result]
```



```python
#add##########################################################################################################
k_means = cluster.KMeans(n_clusters=4, halflife=0.4, sigma=3, seed=0)
user_type_new = None
feature_dim = 2  # coin, enemy destroy, step, shotcnt
n_cluster = 4
old_coin = 0
old_kill = 0
frame_cnt = 0
dict_collector = {'Hardcore Killer': 0, 'Hardcore Archiever': 0, 'Casual Killer': 0, 'Casual Archiever': 0}
##############################################################################################################

# --------------------
# Main Game Play Loop
# --------------------
```



```python
frame_cnt += 1
frame_interval = 240
if frame_cnt%frame_interval == 0:
    #add#####################################################################################
    #user_type_new
    coin_increase = (coin_count - old_coin)#/frame_interval
    kill_increase = (destroyed_enemy_count - old_kill)#/frame_interval หาร lv ดีไหม
    cluster_feature = [coin_increase, kill_increase]  # walking_accuracy, shooting_accuracy
    if coin_count == 0 and destroyed_enemy_count == 0:
        user_type_new = None
    else:
        if coin_count != old_coin or destroyed_enemy_count != old_kill:
            # convert input from list to dict
            feature_dict = convert_list_to_dict(cluster_feature)
            print(feature_dict)
            # predict clustering input
            kmeans_result = k_means.predict_one(feature_dict)
            # clustream_result = clustream.predict_one(feature_dict)
            print(k_means.centers)
            print(kmeans_result)
            gamer_type_k = gemer_type_finder(n_cluster, k_means.centers, kmeans_result)
            # gamer_type_clu = gemer_type_finder(n_cluster, clustream.centers, clustream_result)
            print(gamer_type_k)
            dict_collector[gamer_type_k] += 1

            # update clustering with input
            k_means.learn_one(feature_dict)
            # clustream.learn_one(feature_dict)
            user_type_new = gamer_type_k
            old_coin = coin_count
            old_kill = destroyed_enemy_count
    frame_cnt = 0
#############################################################################################
```



Thanks!

Do you have any questions?