Yapay Zeka

Projesi

Reinforcement Learning Tabanlı Strateji Oyunu

YUSUF ANI - 16011033
AHMET AYDIN - 16011003

2020

İçindekiler

1-) Giriş

Günümüzde hemen hemen her alanda yapay zeka modellerinin etkilerini görmekteyiz. Projemizde yapay zekanın alt alanlarından olan Reinforcement Learning'i bir strateji oyununda uygulayarak hangi etkenlerin daha başarılı modeller oluşturduğunu ve ne gibi kısıtlarımızın olabileceğini araştırıldı. Projede Reinforcement Learning'in uygulanması için popüler yaklaşımlar olan Q-learning ve Deep Q Learning kullanılmıştır.

Projede sonucunda oluşan kodlarda sadece şablon olarak linkteki[1] kodlar kullanılmıştır. Modelin ortamının tasarımı ve kodlanması ve networkünün tasarımı ve kodlanması bize aittir.

Projenin 4 farklı alt modülü vardır. Bu modüller arasına oyunun tasarımı ve networkün tasarımı değişmektedir. Her modülün altında modülde kullanılan oyunun tasarımı hakkında bilgi verilmiştir. Genel olarak bilgi vermek gerekirse oyunumuzda Ajan olarak adlandırdığımız bir veya daha fazla yapay zeka modeli/modelleri karşısında belirli algoritmalarla yapay zeka olmadan oynayan bir Düşman oyuncusu bulunmaktadır. Rapor boyunca Ajan ve Düşman taraflarının unutulmaması önem arz etmektedir. Ayrıca her ikisininde de birer Kale bulunmaktadır. Bu kaleler yok edilmediği sürece oyuncular tekrar kalelere yakın lokasyonlarda canlanabilmektedir. Her modülde farklı boyutlarda kare bir harita kullanılmıştır.

Sistemin başarısını ölçmek için ödül/ceza sistemi kullanılmıştır. Ceza aslında ayrı bir değer olarak değil ödül değerinin negatif olduğu durumları temsil etmek için kullanılmıştır. Ajan yaptığı belirli aksiyonlara karşı oyunun kuralları gereğince belirli ödül veya cezalar almıştır. Ödül ve ceza değerlerini optimize etmek de büyük bir problem olmuştur. Her modülde İlgili ödül ve ceza değerleri bulunmaktadır.

Not olarak eklenmelidir ki proje yaklaşık 3 hafta sürmüştür. Elimizde yaklaşık 200'den fazla model ve çok fazla sayıda log dosyası vardır. Bu durumdan dolayı ilgili modüllerde yapılan işlemlerde genel olarak ne amaçlandığı ve sonuçlarından bahsedilecektir.

Projenin çalıştırılması için requirements.txt dosyasındaki kütüphanelerin yüklü olması zorunludur.

2 - Modül -1

İlk modelimizde amacımız Q Learning kullanarak bir Düşman oyuncusuna karşı başarı elde etmektir. Q Learning için gerekli Q-Tablo oluşturulmuş Bir bakıma yapılacak proje için fizibilite aşaması olmuştur.

2.1 – Ortam Tasarımı

Q-Tablo oluşturulması için oyuncular ve düşman Kalenin x ve y koordinatları durumlar olarak alınmıştır. Bu durumda x ve y değerleri harita büyüklüğüne bağlı olmaktadır. Oluşturulan Q-Tablo bellekte saklandığı için çok büyük harita büyüklüğünün seçilemeyeceğini görülmüştür. Kendi cihazımızda

10 *10 büyüklüğünde bir harita için neredeyse yarım gün haritanın oluşturulması gerektiği görülmüş ve denemelerde 4 , 5 , 6 harita büyüklük değerleri denenmiştir. Ayrıca buradan şu yorum çıkartılabilir ki Q – Learning gerçek hayatta hatta karmaşık durumlarda bile gereksinimleri dolayısıyla etkili bir yöntem olarak gözükmemektedir.

Ortamda 1 adet Ajan oyuncusu ve 1 adet Düşman oyuncusu kullanılmıştır. Düşman oyuncusu yönlendirilmesi için Düşman oyuncusu ile Ajan oyuncusu ve kalesinin koordinatları arasındaki fark alınmıştır. Çıkan sonuçtan küçük olana gidecek şekilde bir yönlendirme sağlanmıştır.

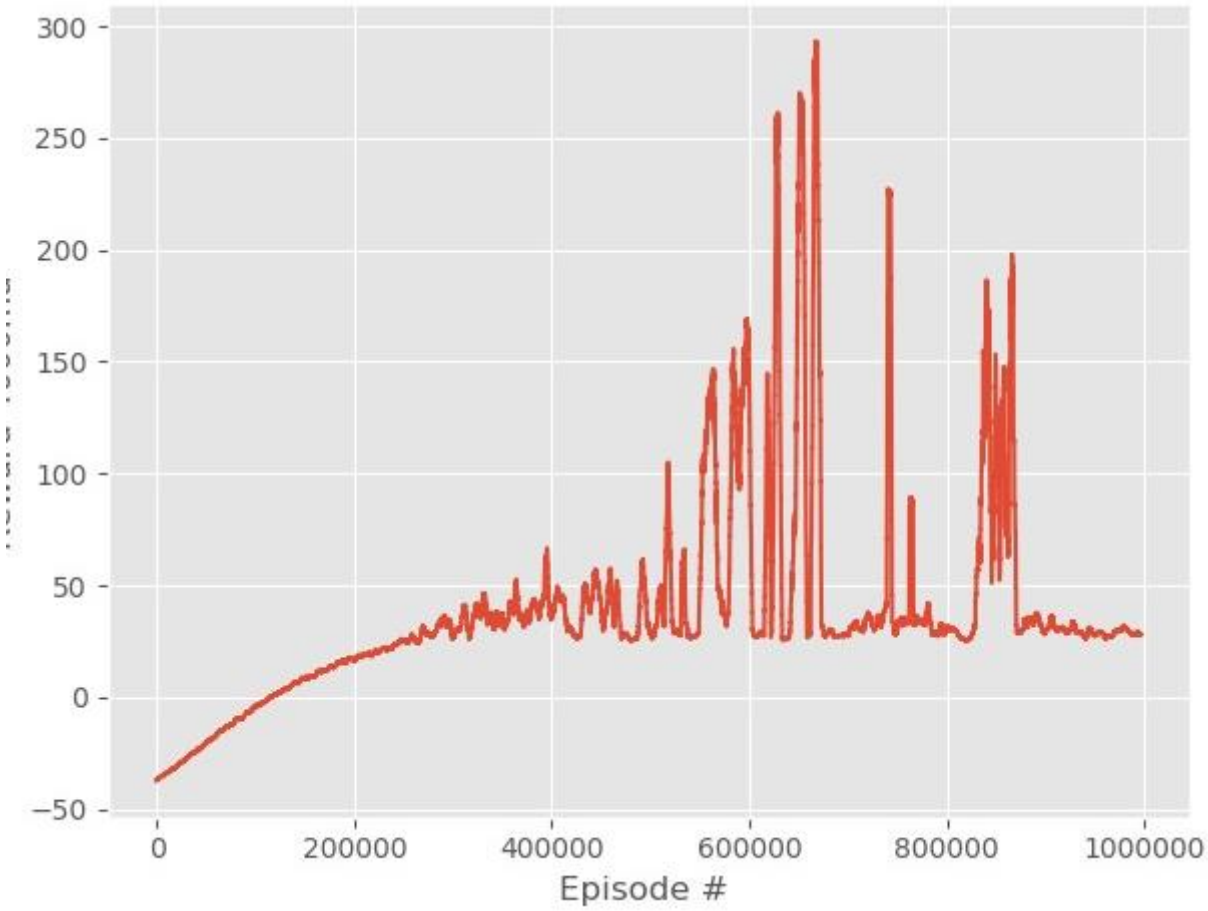
Ajan oyuncusunun yönetilmesi için Q-Tablo değerleri ile eğitimin başlarında epsilon greedy yöntemi kullanılmıştır. Burada başlarda epsilonun azalma katsayısını 0.975 yerinde 0,99 değerine çekmek daha başarılı sonuçlar vermektedir. Bunun nedeni ise haritanın küçük olmasından dolayı oyuncunun çabuk oyunu kaybedip çok fazla ulaşamadığı Q değerinin olması olarak yorumlanabilir.

Yukarıda bahsettiğim gibi karşıdaki modelin hazır olup iyi oynaması sebebiyle Ajan oyuncusu eğitimi için gerekli adım sayısını bulamadığı fark edilmiştir. Bu sorun proje boyunca en büyük sorunlardan biri olmuştur. Düşman modeli çok güçlü olması eğitimi imkansız hale getirirken , çok güçsüz olması durumunda ise model çok çabuk öldürmeyi öğrendiği için proje amacına ulaşmış olarak kabul edilmedi. Bu modüldeki çözümümüz ise Düşman oyuncusunun adımlarını bir sabitle kısıtlamak oldu . Rakip oyuncu her X adımda bir hareket ettirildi. X değeri bizim denemelerimizde 5 olarak denendi.

Ortam değerleri ve Ödüller:

```
SIZE = 5 # Harita Size
EPISODES= 1000000 # Number of episode

KALE_HEALTH= 10
PLAYER_ATTACK = 2 # Player attack power

MOVE_PENALTY = 1 # Ajan player get penalty for each step
DÜŞMAN_KALE_HIT_PENALTY = 5 # If Düşman player hit to Ajan Kale
DÜŞMAN_PLAYER_HIT_PENALTY = 2 # If  player  hit to Ajan
KALE_HIT_PENALTY = 5 # If AJAN hit to ENENMY Kale
PLAYER_HIT_PENALTY = 2 # If AJAN hit to Düşman player
WIN_REWARD = 10 # LOSE REWARD = -1 * WIN REWARD

epsilon = 1
EPS_DECAY = 0.99999 # Epsilon greedy
SHOW_EVERY = 4000 # Show Game in every SHOW_EVERY STEP

LEARNING_RATE = 0.1
DISCOUNT = 0.95

AJAN_N = 1 # Number of Ajan
DÜŞMAN_N = 1 #Number of Düşman
```

2-1 Sonuç



Yukarıdaki grafik x ekseninde episode numarasını ve y ekseninde ise ödül değerlerini göstermektedir. Grafiğe bakılarak kazanma ödülünün 10 ve kaybetme ödülünün -10 olduğu durumlarda bu kadar yüksek ödül alması bizi de şaşırtmıştır. Modeli deneyimleyince modelin daha fazla puan almak için rakibi öldürmek yerine oyuncusunu sürekli öldürerek 2'şer puan kazanmayı tercih etmiştir. Bu da bizim düşünmediğimiz bir durumdu :D

2-2 Modül-2

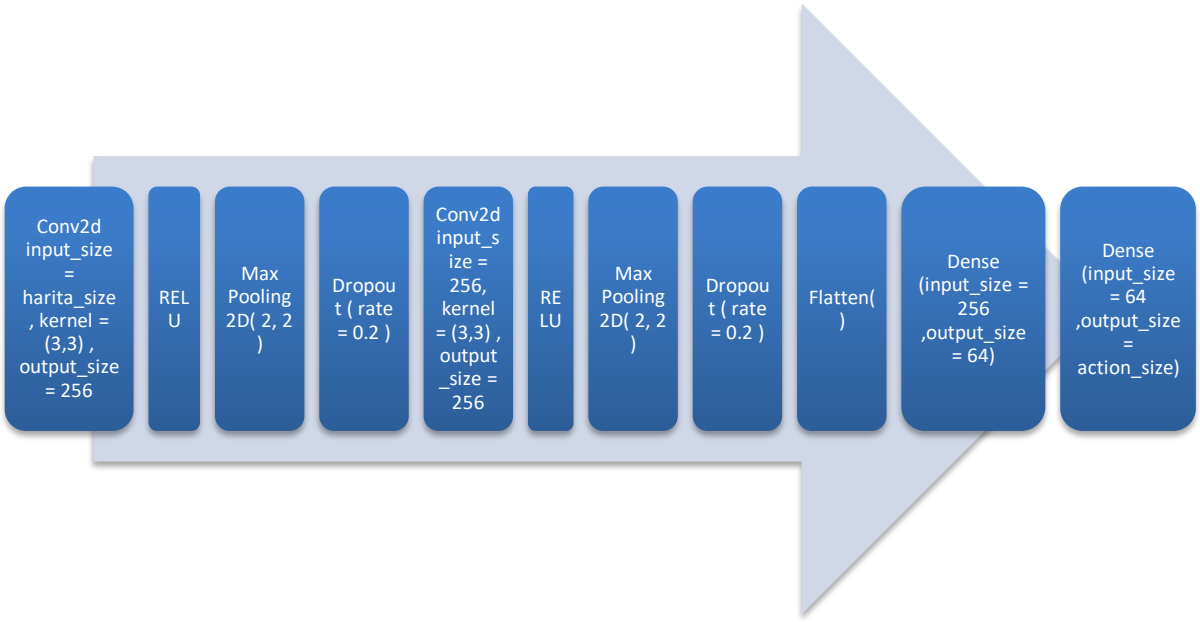
Her ne kadar Q learning güzel sonuçlar vermiş olsa da işi biraz daha karmaşık hale getirmek istedik. Bunun için tasarım kısmında çok fazla değişiklik gerekti.

2.2.1- Ortam Tasarımı

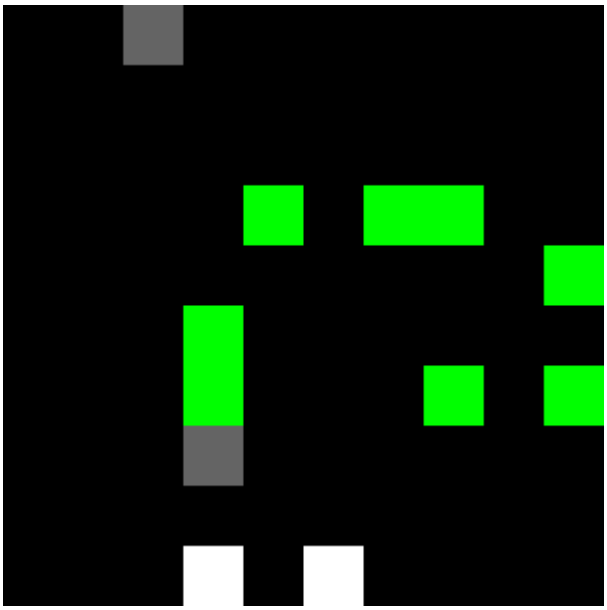
Bunun için öncelikli amacımız Deep Q-Learning algoritmasını kodumuza eklemek oldu. Bunun için bize birkaç Python yapısı haricinde bir Yapay sinir ağı yapısı gerekti. Bunun için kullanımı en kolay olan

Keras kütüphanesini tercih ettik. Kodun eğitilirken eğitim bilgilerini görebilmek amacıyla da Tensorboard kütüphanesini kullandık.

Yapay sinir ağı yapısını CNN mimarisi üzerine kurduk. Kısaca giriş katmanında modelin alacağı resim boyutu kadar bir girdi alan ve aksiyon sayısı kadar sonuç üreten bir model oluşturuldu. Modelin iç yapısı :



Yapay sinir ağına verilen harita resmi için renklerden oluşan bir temsil gerekiyordu. Bunun için şu şekilde bir resim oluşturduk:



Burada beyazlar ajan oyuncusunu ve kalesini, griler düşman oyuncusunu ve kalesini simgelerken , yeşiller ise bir sonraki paragrafta bahsedeceğimiz ormanlık alanı temsil ediyor. Ayrıca ayarlardan kaç tane ağaç olacağı ayarlanabiliyor ve bu ağaçlar rastgele olarak oluşuyorlar. Bu da modelin yolu ezberlemesinin önüne geçiyor.

Oyunu biraz daha karmaşık ve ajanımızın daha stratejik düşünmesi amacıyla ormanlık alan eklemeye karar verdik. Eğer Ajan ormanlık alana gitmeye çalıştıysa belirlenene ceza puanını verildi.

Oyuncu sınıfı ve kale sınıfı ile ilgili değişiklikler yapıldı. Oyuncular kalelerin önünde rastgele bir konumda canlanıyorlar. Aynı şekilde Kale'ler de rastgele şekilde konumlandırılıyorlar. Oyuncular için aksiyon sayısı 9'a çıktı. Bu aksiyonlar yukarı , aşağı , sağ , sol , sabit kalmak ve çapraz yönler olarak sıralanmıştır. Ayrıca düşman oyuncu için de yol bulma algoritmasının gelişmesi gerektiği farkına varılmıştır çünkü Öklid olarak uzaklığa giderken düşmanın da önüne ağaçlar çıkabilir ve oyunun gerçekçiliği bozulur. Bu yol bulma algoritması için Breath First Search seçilmiştir. BFS algoritmasını seçmemizin sebebi uygulaması en kolay algoritmalardan biri olması çünkü zaman anlamında kısıtımız vardı. Belki bu algoritma ileride A* ile değiştirilebilir. Algoritma çalışamadan önce yine en yakın birim bulunuyor ardından o birim için bir BFS algoritması çalıştırılıyor. Ardından bulunan path'den geri gelinerek bir sonraki adım bulunuyor . Buna göre Düşman oyuncusu yönlendiriliyor.

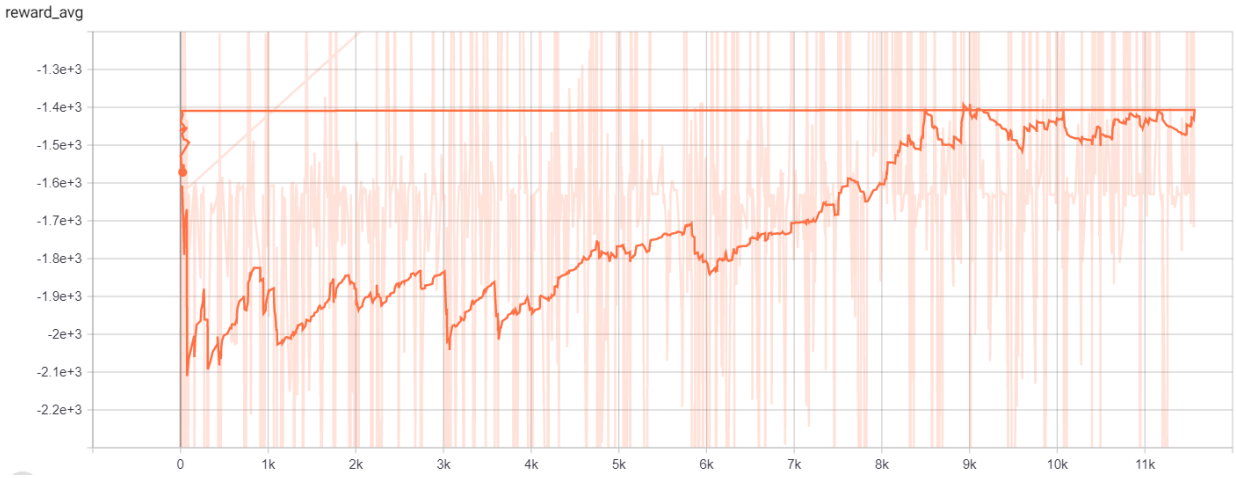
Ortam değerleri ve Ödüller:

```
SIZE = 10 # HARİTA SIZE
OBSERVATION_SPACE_VALUES = (SIZE, SIZE, 3),  # 4 ,
ACTION_SPACE_SIZE = 9 # Number of Action
RETURN_IMAGES = True #
MOVE_PENALTY = -1 #
FOREST_PENALTY = -10 # If the Ajan goes to the tree
p = {
    "FOREST_AREA" : 8 ,

    "AJAN_TO": {"DÜŞMAN_PLAYER": 25, "DÜŞMAN_KALE": 100}, # rewards for If Ajan
attack Düşman
    "DÜŞMAN_TO": {"AJAN_PLAYER": -25, "AJAN_KALE": -100}, # rewards for If Düşman
attack Ajan

    "LOSS": -500,
    "WIN": 500,

    "KALE_LEN": 1,

    "AJAN_PLAYER_HEALTH": 100,
    "AJAN_KALE_HEALTH": 1000,
    "AJAN_PLAYER_ATTACK_POINT": 150,

    "DÜŞMAN_PLAYER_HEALTH": 100,
    "DÜŞMAN_KALE_HEALTH": 1000,
    "DÜŞMAN_PLAYER_ATTACK_POINT": 50
}
```

```python
# HYPERPAREMETERS
# Environment settings
EPISODES = 20_000
MIN_REWARD = -200  # For model save
# Exploration settings

epsilon = 1 if model_path=="" else 0.1
#epsilon = 1
# not a constant, going to be decayed
EPSILON_DECAY = 0.99975
MIN_EPSILON = 0.001

#  Stats settings
AGGREGATE_STATS_EVERY = 1  # episodes
SHOW_PREVIEW = True



env = Env(SHOW_PREVIEW)

# For stats
ep_rewards = [-200]

# For more repetitive results
random.seed(1)
np.random.seed(1)
tf.set_random_seed(1)
```

Burada harita büyüdüğü için epsilonu eski değerine geri getirdik. Ayrıca bundan sonraki modellerin aynı başlangıç noktalarından başlatarak daha karşılaştırılabilir sonuçlar elde ettik.

2.2.2 – Sonuç



Yukarıdaki grafikte alınan ödül miktarında bir artış gözüküyor. Bu grafik için bir smoothing uygulanmıştır. Burada maalesef eğitimler için Google Colab ortamına geçtiğimiz için 11k ya kadar ki kısım kaydedilmiş. Biz modeli 20 k adım eğittik ve modelin başarısı arttı .

Modeli test ettiğimizde modelin karşı rakibin geldiği yeri tahmin ederek onu öldürerek puan kazandığını gördük. Aslında model amacımızın dışına çıktı fakat yine de oyun kuralları içerisinde kaldığı için başarılı olarak gördük.

2.3 – Modül-3

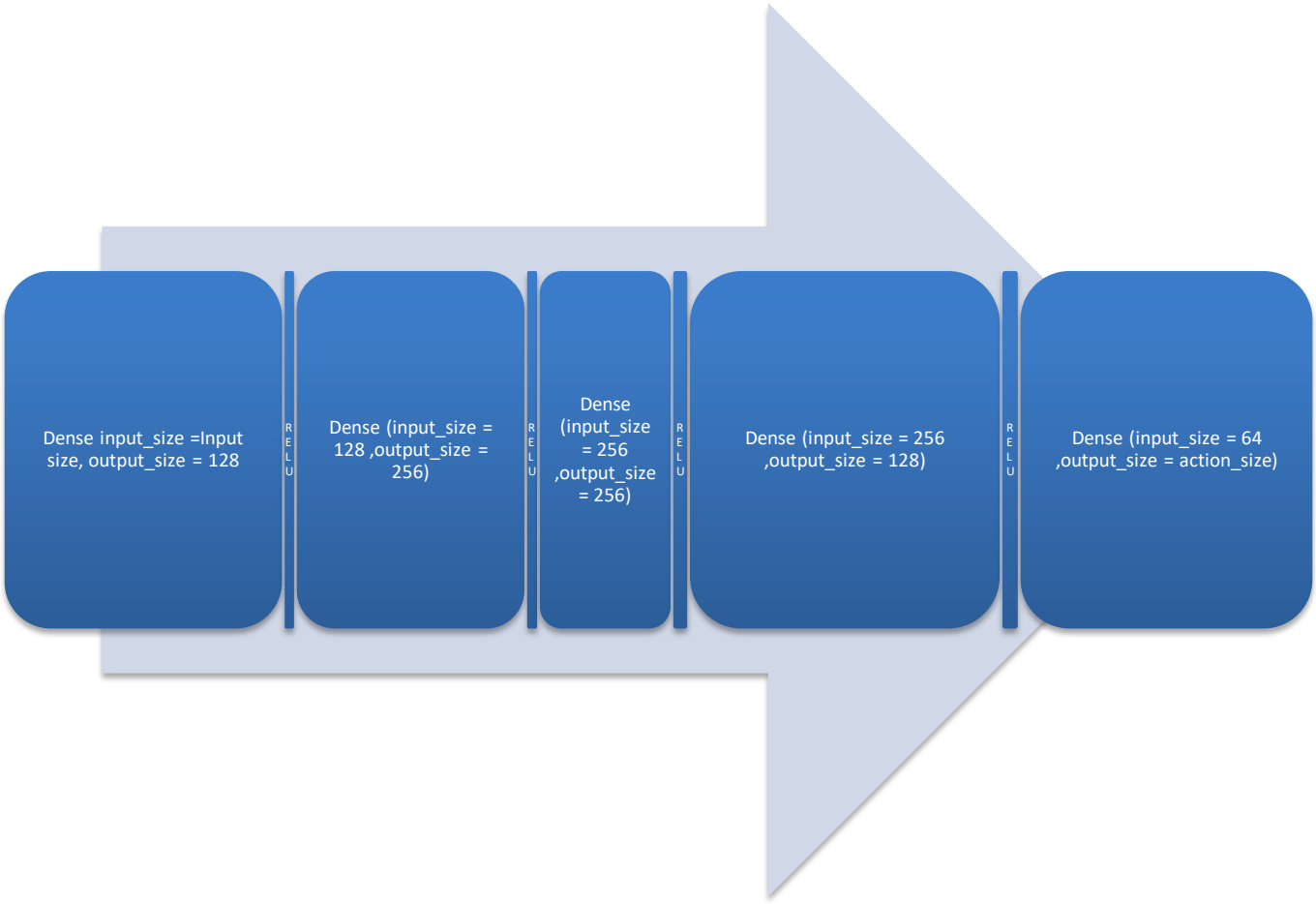
Bu modülde işi bir adım öteye taşımaya karar verdik. Yapmak istediğimiz şey bir ajanın birden fazla oyuncuyu aynı anda kontrol etmesi. Bu durum ise en çok zamanımızı harcayan durum oldu.

2.3.1- Ortam Tasarımı

Bu modelde önceki modelin yakaladığı oyun ile ilgili hataları düzeltmekten ve networkümüzü değiştirmek haricinde büyük değişiklikler yapmadık. Oyunu görselleştirmek için Pygame kütüphanesi ve internetten bedava simgeler bulduk . Eski yaklaşımla beraber Network konusunda da yeni bir yaklaşım kullandık: İnternette araştırdığımızda çoğu Reinforcement Learning CNN mimarileri üzerinde kurulmuştu. Bunun nedeni ise genelde Reinforcement Learning ile yapılmış oyunlarda insanla bir karşılaştırma içine giriliyor. İnsanlar durumları gözetmek için sadece göz ile veri alabildiği için modeli için de aynı durum düşünülmüş. Bizim oyunumuzda insanın oynaması için şimdilik bir ara yüz yok. Bu yüzden biz Fully Connected Layerlarla da sonuç alabileceğimizi düşündük.

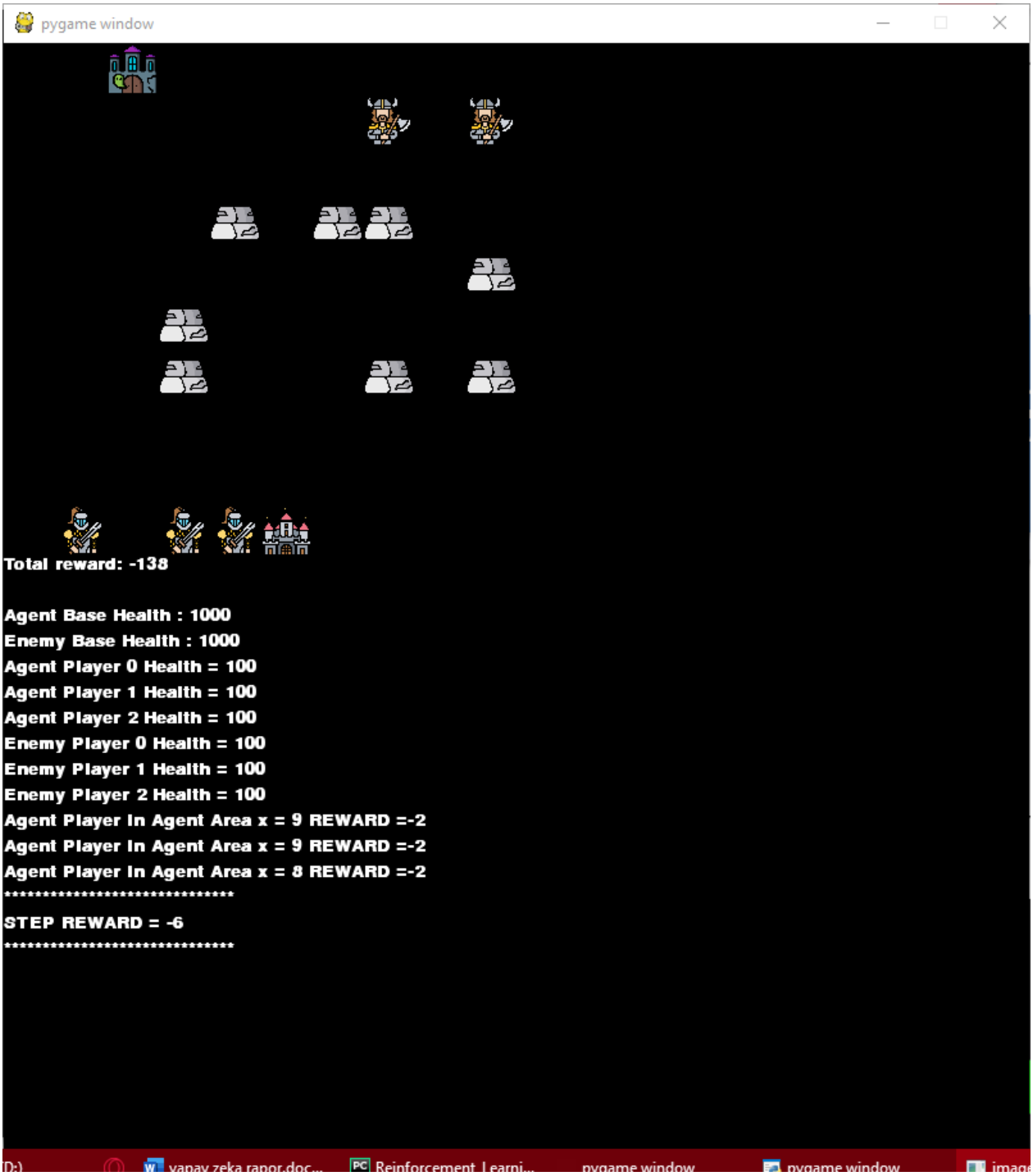
Yeni networkümüzün yapısı şu şekilde oldu :

Input Size : Harita + Oyuncuların Birbirilerine uzaklıkları + Oyuncuların ve Kalelerin canları

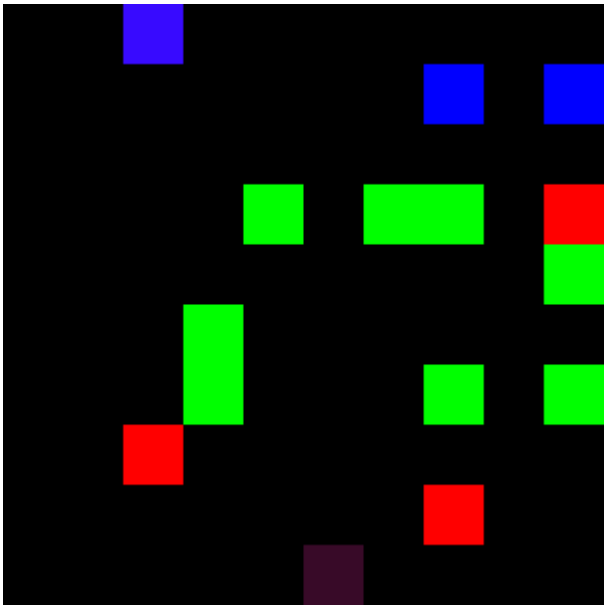


Output olarak ise (Ajanın yönettiği oyuncu sayısı * Her bir oyuncunun aksiyon sayısı) aldık . 3 Ajanlı 9 aksiyonlu bir oyun için 27 observation değeri olmuş olur .Q değerlerinin güncellemek için ise ilk Ajan için ilk 9 değeri alıp maksimum olanı seçilip aksiyon olarak verildi. Yeni durum modele tekrar verilip gelen değerler arasında ilk 9'u alınarak Deep Q learning denklemi ile değer hesaplandı. İkinci Ajan ve üçüncü Ajan için ise kendilerine denk gelen sonuçlar alınıp değerler güncellendi.

Bunlara ek olarak CNN modeli ve Fully Connected Layerlar ayrı ayrı denenerek performans karşılaştırması yapıldı.



Arayüz örneği



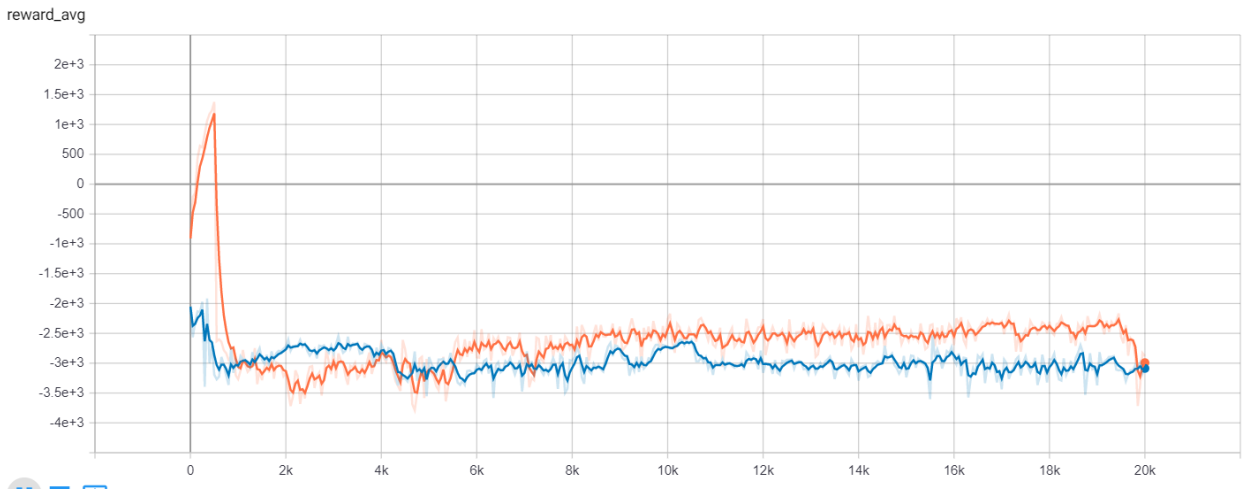
Modelin aldığı Örnek input

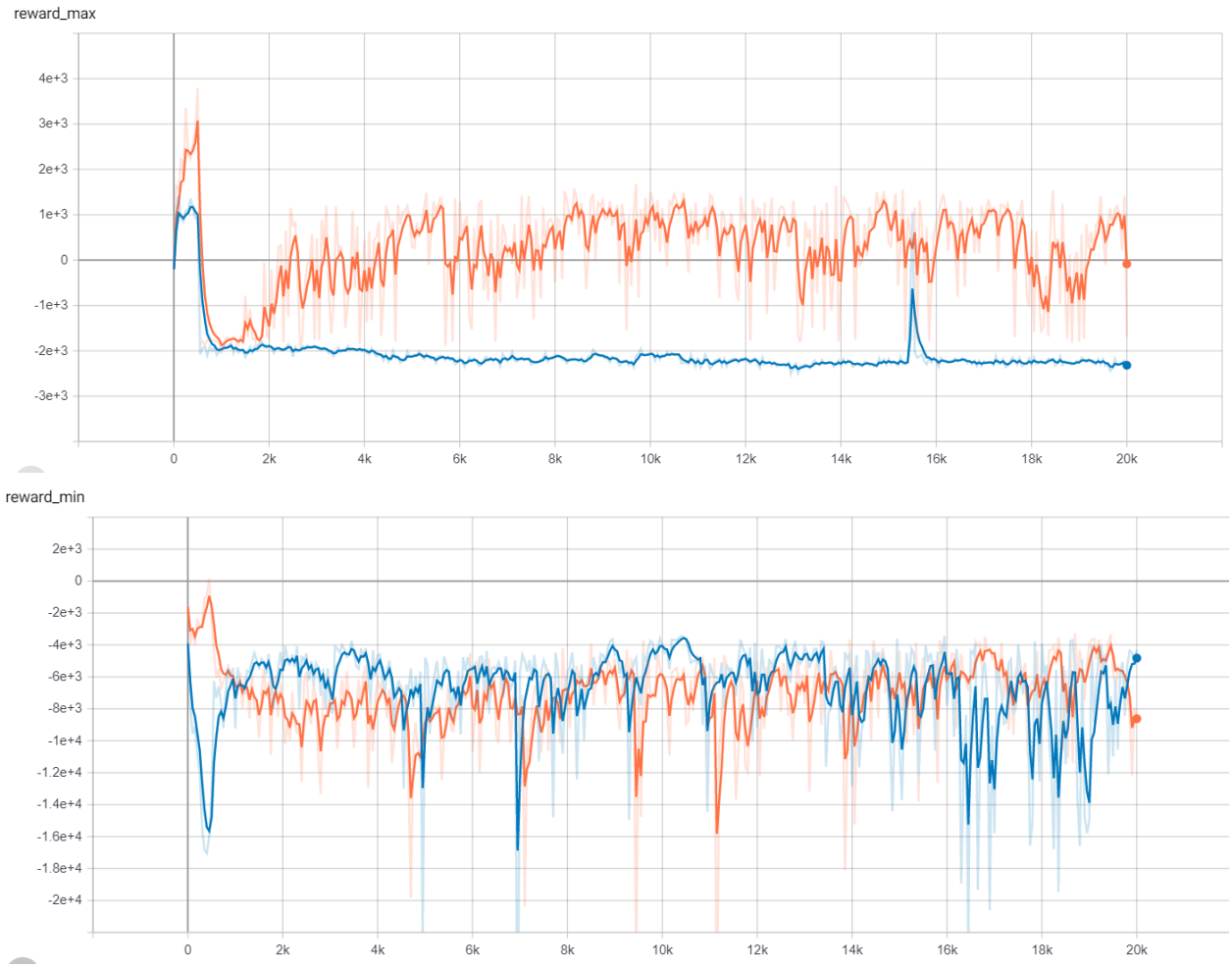
Ortam değerleri ve Ödüller:

```
NUMBER_OF_AJAN_PLAYER = 3
NUMBER_OF_DÜŞMAN_PLAYER = 3
SIZE = 10
#OBSERVATION_SPACE_VALUES =
SIZE*SIZE+(NUMBER_OF_AJAN_PLAYER+1)*(NUMBER_OF_DÜŞMAN_PLAYER+1)+NUMBER_OF_AJAN_PLAYER
+NUMBER_OF_DÜŞMAN_PLAYER+2  # +2 for Kale health
OBSERVATION_SPACE_VALUES = (SIZE,SIZE,3)

ACTION_SPACE_SIZE = 9

MOVE_PENALTY_AJAN_AREA = -2
MOVE_PENALTY_DÜŞMAN_AREA = 1

FOREST_PENALTY = -20
p = {
    "FOREST_AREA" : 8 ,

    "AJAN_TO": {"DÜŞMAN_PLAYER": 40, "DÜŞMAN_KALE": 80},
    "DÜŞMAN_TO": {"AJAN_PLAYER": -20, "AJAN_KALE": -50},

    "LOSS": -500,
    "WIN": 1000,

    "PLAYER_N": 1,  # player key in dict
    "FOOD_N": 2,  # food key in dict
    "DÜŞMAN_N": 3,  # Düşman key in dict
    "KALE_LEN": 1,

    "AJAN_PLAYER_HEALTH": 100,
    "AJAN_KALE_HEALTH": 1000,
```

```
    "AJAN_PLAYER_ATTACK_POINT": 150,

    "DÜŞMAN_PLAYER_HEALTH": 100,
    "DÜŞMAN_KALE_HEALTH": 1000,
    "DÜŞMAN_PLAYER_ATTACK_POINT": 50
}
```

2.3.2- Sonuç





Yukarıdaki grafiklerde 5şer adımın sırıyla ortalama ödül , maksimum ödül ve minimum ödül değerleri gözükmektedir. Mavi çizgi CNN modelini temsil ederken turuncu renk temsil etmektedir.

Öncelikle modeller arasında çok fark olmadığı gözükmüştür fakat biraz da olsun Fully connected layerlı modelin daha önde olduğu gözüküyor. Ayrıca grafikler incelendiğinde modellerin başarı oranlarında büyük değişiklikler olmadığı gözüküyor. Bu yüzden Modül -3 denemeler sonucunda başarısız olmuştur.

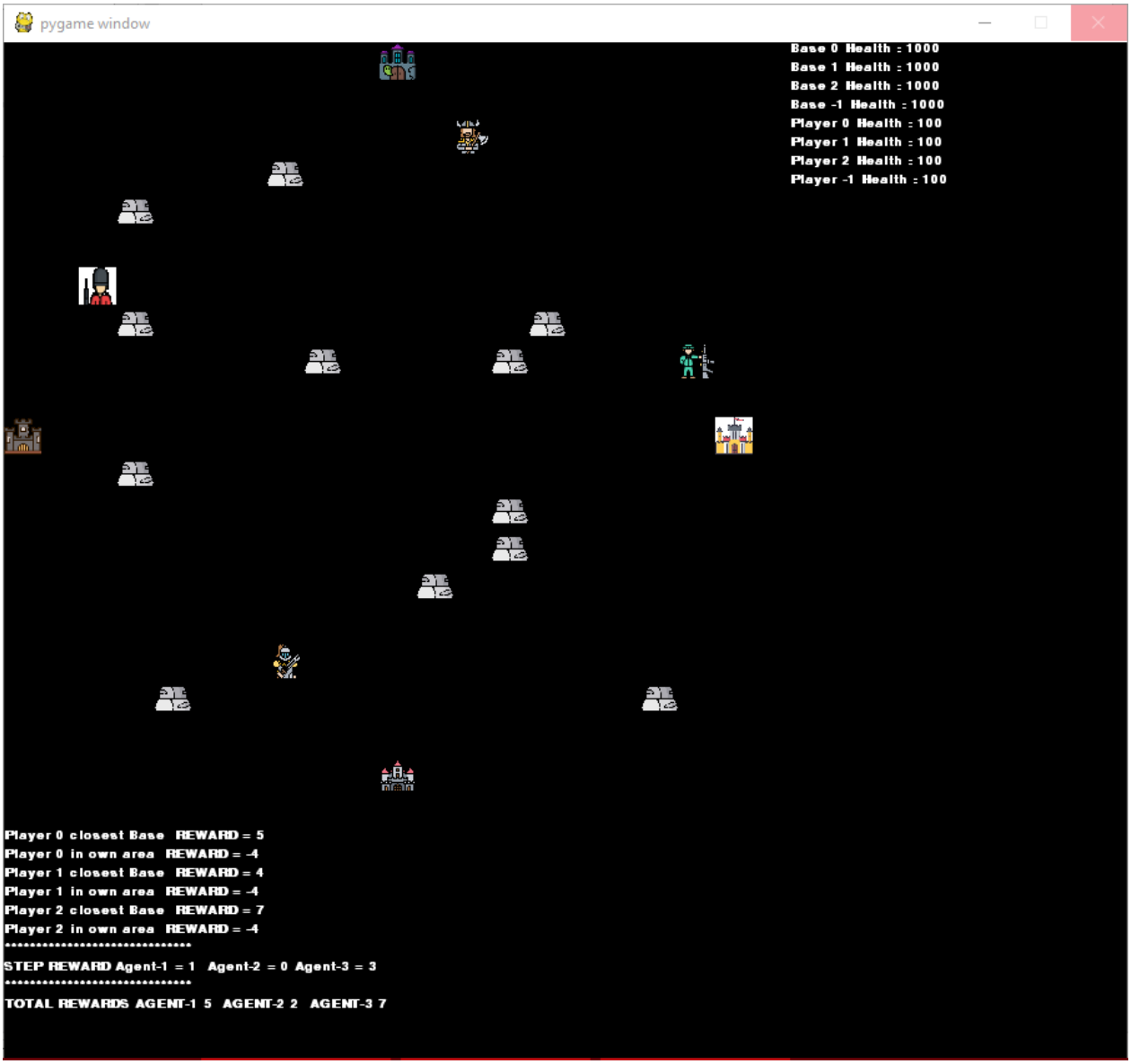
Başarısızlığın nedenini araştırınca multi agent Reinforcement Learningin daha farklı uygulandığını başlı başına bir konu olduğunu ve üstünde çalışmaların devam ettiğini gözlemledik.

2.4 Modül -4

Buraya kadar olan modellerle ödev için kriterleri sağladığımızı düşünüyoruz fakat aklımıza diğer modellerden çok daha eğlenceli ve farklı bir yapı denemeye karar verdik.

2.4.1 – Ortam Tasarımı

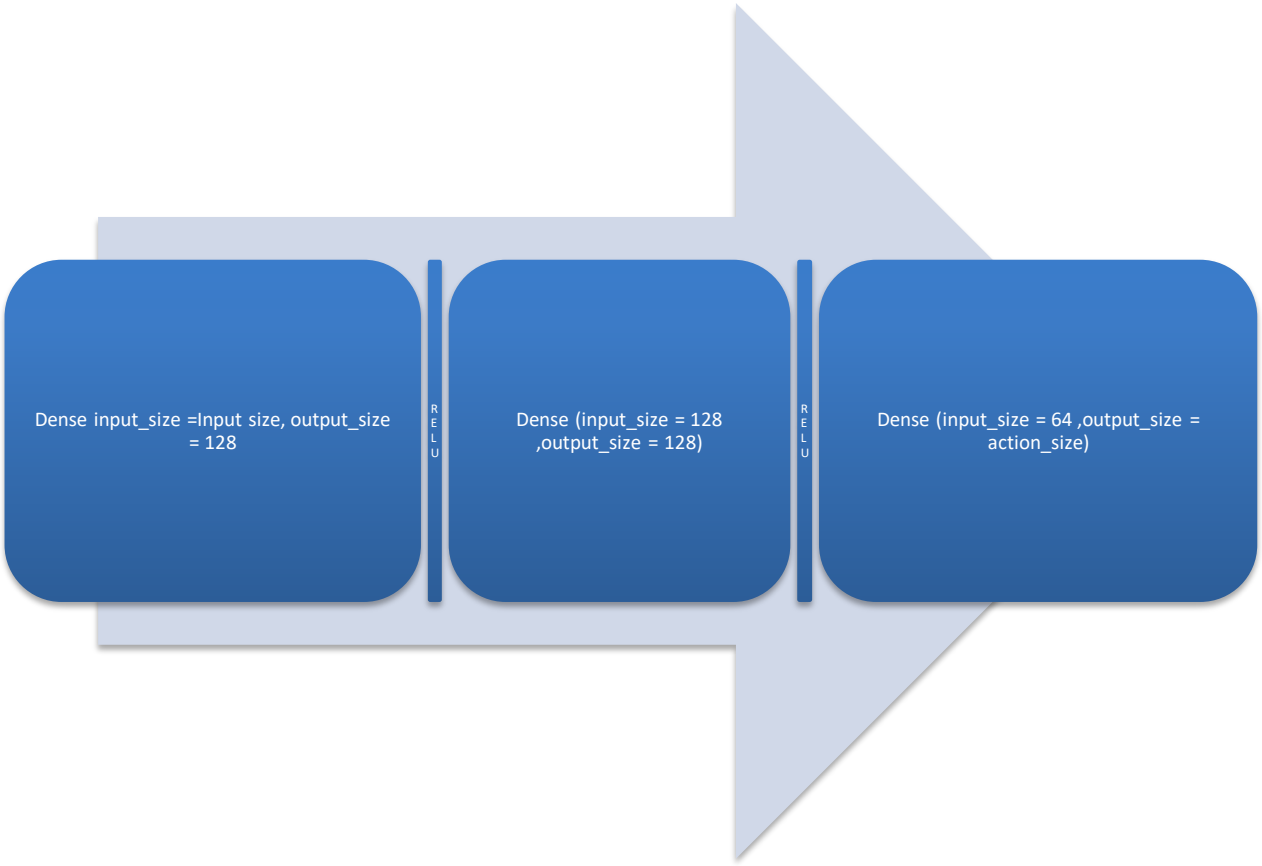
Burada aslında 2. Modülün çok benzerini yaptık. Farklı olarak elimizde 1 tane Ajan modeli yerine 3 tane Ajan modeli var ve ortak olarak Düşman modeli bulunuyor. Ayrıca Yapay sinir ağı yapısında da biraz daha performanslı olduğuna inandığımız Fully Connected Layerlarla oluşturduk. Ayrıca modellere daha fazla alan vermek için 10 olan harita boyutunu 20'ye çıkardım. Ayrıca son kullanıma hazırlamak üzere ara yüzü ve bilgilendirme ekranlarını düzelttik .



Yapay sinir ağı yapısı :

Input : (Harita Size * Harita Size + İlgili Ajanın diğer tüm Ajan ve Düşman birimlerine uzaklıkları + Tüm Ajan ve Düşman birimlerinin canları)

Output : 8 Adet aksiyon (aksiyonlar yukarı , aşağı , sağ , sol ve çapraz yönler olarak sıralanmıştır)



Ortam değerleri ve Ödüller:

Önceki modellerden farklı olarak daha fazla saldırıya yatkın modeller yapmak için ödül değerlerinde oynamalar gerçekleştirdik. Bu oynamaları aşağıdaki belirtilmiştir:

Modellere yeni olarak :

- Harita dışına çıkması denemesi halinde -10 puan ceza verildi.
- Modele en yakın kalenin Ajan oyuncusuna ulan uzaklığı ile orantılı bir ödül sistemi kuruldu:
- Ödül = Harita boyutu / 2*uzaklık
- Eğer Ajan oyuncusu kaleye yakın ise yüksek başarı elde edecektir.
- Artık ağaçların koordinatları kullanıcıdan alınabiliyor.

2.4.2 – Sonuçlar

Modelleri başarını ölçmek için ödül değerlerini göstermek pek uygun olmayacağını düşündük. Çünkü ortam sabit olmayacağı için modellerin başarısı karşıdaki modellerin başarısı ile ters orantılı olduğunu gördük.

Sonuçlardan kesin olarak çıkarabileceğimiz konuş ise modellerin hepsinin başarısının zamanla artması oldu.

Bir başka gözlemimiz ise modelleri sadece 2500 bölüm eğitmemiz oldu. Önceki modellerde de bu değer 20000'di. Bunun da en büyük sebebi oyunun bitmesine kadar modellerin daha fazla zamanı olması olarak görülebilir.

Modellerdeki arasında if komutlarıyla yönetilen modelle en çok karşılaşanın en çok başarıyı aldığını gördük.

3- Genel Sonuçlar

1. Biz çalışmalarımızda Deep Q learning kullandık. Hazır bir aksiyon , araba yarışı veya basit labirent tarzı oyundaki performansının çok iyi olduğu biliniyor. Hazır bir takım yapay sinir ağları ile uygulanabilirliğini gördük. Fakat şu bir gerçek ki strateji gibi karmaşık oyunlarda Deep Q Learning'in üstüne başka algoritmalar da eklenmesi buna göre yapay sinir ağı tasarlanması gerekebilir. Yine de küçük ölçek için öğrenebilme yetisinin olduğunu gördük.
2. Ödüllerin ne olduğu ve ne kadar olduğu çok fazla fark ediyor. Çoğu durumda modelin bir ödülü almanın kısa yolunu bulup ezberlediğini görüldü.
3. Reinforcement Learning Ajanlarının oyunlardaki hataları bulup faydalanması diğer oyuncuların bulmasına oranla çok daha hızlı oluyor. Bug bulma alanında kullanılabilir.
4. Projedeki gibi karşıda bir rakibin olduğu durumlarda rakibin hamleleri çok fazla etki ediyor. Rakip ne çok güçlü olmalı ne de çok güçsüz olmalı.

4- Kaynakça

1- https://pythonprogbellekming.net/training-deep-q-learning-dqn-reinforcement-learning-python-tutorial/