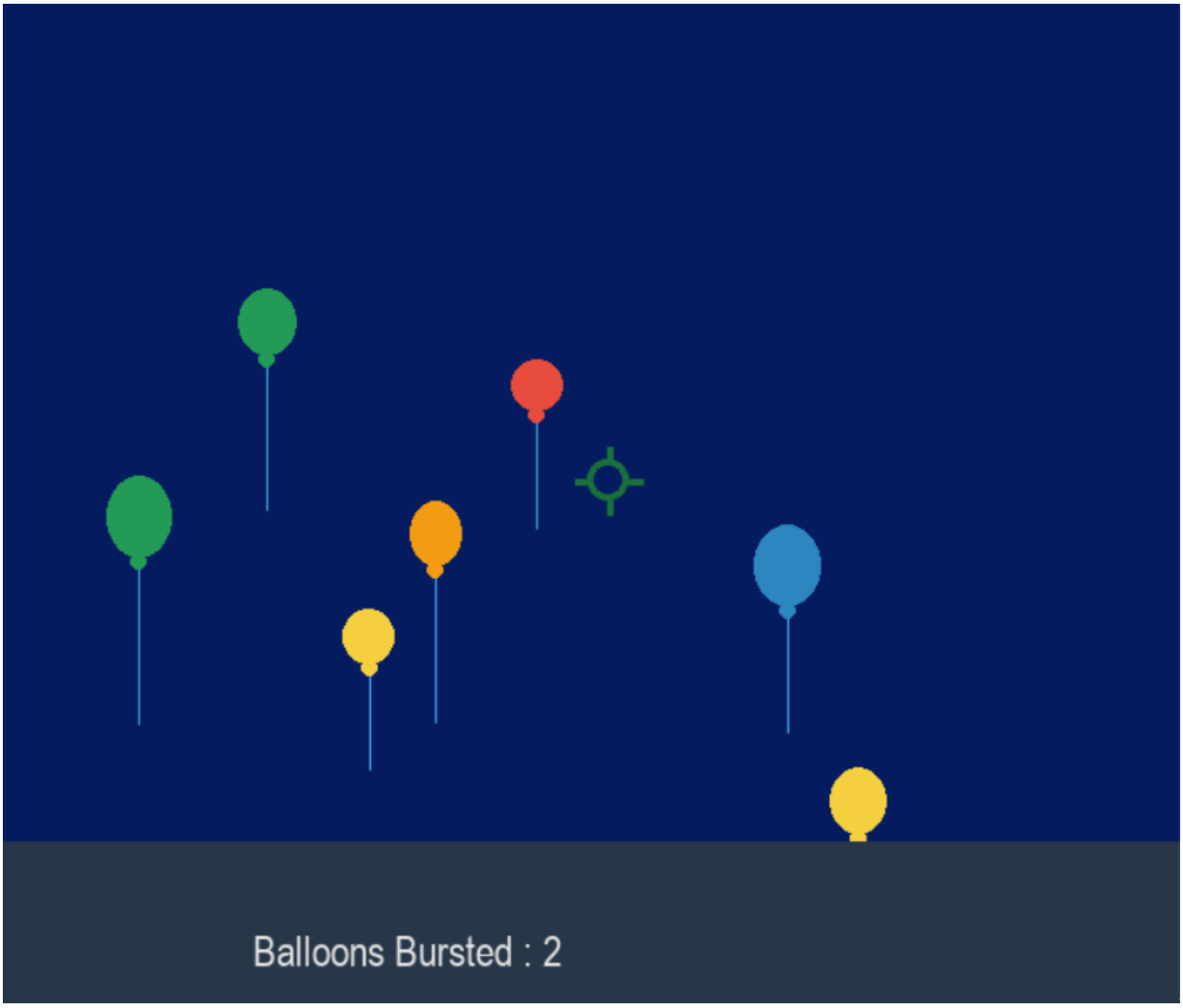
YAPAY ZEKAYA GİRİŞ DÖNEM PROJESİ

Ahmet Enis ŞİMŞİR – 19011077

Kutay ALPTEKİN – 20011615

Projemizde amaç, pygame kütüphanesi ile kodlanmış bir oyunu oynamayı öğrenen bir ajan oluşturmaktır. Ajanın öğrenme sürecini DQN yöntemi ile sağladık.

Oyun olarak ise bir adet balon patlama oyununu kullandık. Oyundaki tek amaç ekrandaki balonlara fare imleci ile tıklayarak puan kazanmaktır.



Görüldüğü gibi ekranda bizim seçtiğimiz miktarda (bu örnekte 10 adet) balon hareket ediyor ve fare imleci ile üzerlerine tıkladıkça puan kazanıyoruz.

Oyunda yönlendirdiğimiz bir karakter olmadığı için kullandığımız ajanın tek yaptığı şey fare imlecini hareket ettirip ekrana tıklamak.

Bunun için DQN kullandık ve parametrelerini şu şekilde girdik:

```
self.memory = []
self.gamma = 0.9  # Gelecekteki ödülleri ağırlama için indirim faktörü
self.epsilon = 1.0  # Keşfetme ve sömürme dengesi için epsilon değeri
self.epsilon_decay = 0.995  # Her adımda epsilon'un azalma hızı
self.epsilon_min = 0.01  # Minimum epsilon değeri
self.learning_rate = 0.001  # Öğrenme oranı
```

Eylemleri değerlendirme ve deneyimleri hatırlama:

```
def build_model(self):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(24, input_dim=self.state_size, activation='relu'))
    model.add(tf.keras.layers.Dense(24, activation='relu'))
    model.add(tf.keras.layers.Dense(self.action_size, activation='linear'))
    optimizer=tf.keras.optimizers.Adam(lr=self.learning_rate)
    model.compile(optimizer=optimizer,loss='mse')
    return model

def remember(self, state, action, score, next_state, done):
    self.memory.append((state, action, score, next_state, done))
```

Eylem seçme:

```
#eylem seçer
def act(self, state):
    if np.random.rand() <= self.epsilon:
        # Keşfetme: Rastgele bir eylem seç
        return random.randint(0, self.action_size - 1)
    else:
        # Sömürme: En iyi tahmini eylemi seç
        q_values = self.model.predict(state)[0]
        if q_values[0] == np.max(q_values):
            # Yukarıya hareket et
            return 0
        elif q_values[1] == np.max(q_values):
            # Aşağıya hareket et
            return 1
        elif q_values[2] == np.max(q_values):
            # Sola hareket et
            return 2
        else:
            # Sağa hareket et
            return 3
```

Epsilona bağlı bir oranla yeni eylem dener veya geçmişteki en iyi eylemi seçer. Epsilon zamanla azalır ve ajan yeni deneyimlere daha kapalı olur.

Deneyimi kullanarak modeli güncelleme:

```
def replay(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)

    states = []
    targets = []
    for state, action, score, next_state, done in minibatch:
        target = score
        if not done:
            # Q değerini güncelle
            target = (score + self.gamma * np.amax(self.model.predict(next_state)[0]))
        target_f = self.model.predict(state)[0]
        target_f[action] = target

        states.append(state[0])
        targets.append(target_f)

    # Modeli eğit
    self.model.fit(np.array(states), np.array(targets), epochs=1, verbose=0)

# Epsilon değerini azalt
    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay

def load(self, name):
    self.model.load_weights(name)

def save(self, name):
    self.model.save_weights(name)
```

Ayrıca üstte de belirttiğimiz gibi epsilon değerinin zamanla azalması da bu bölümde oluyor. epsilon_decay değişkeni azalma oranını belirliyor. Değişkenin değeri ne kadar

Peki ajan neye göre gelişiyor? Ajanın gelişmesi 2 faktöre bağlı. İlki balon patlatmak.

```
def burst(self):
    global score
    global patlatilan
    pos=pygame.mouse.get_pos()#fare imlecinin yerini alır

    if isonBalloon(self.x, self.y, self.a, self.b, pos):
        score += 20
        patlatilan += 1
        self.reset()
```

Burada ajanımızı ödüllendirmek için kullandığımız **score** değişkeni balon patlayınca 20 artıyor.

İkinci faktör ise balonların yakınında bulunmak.

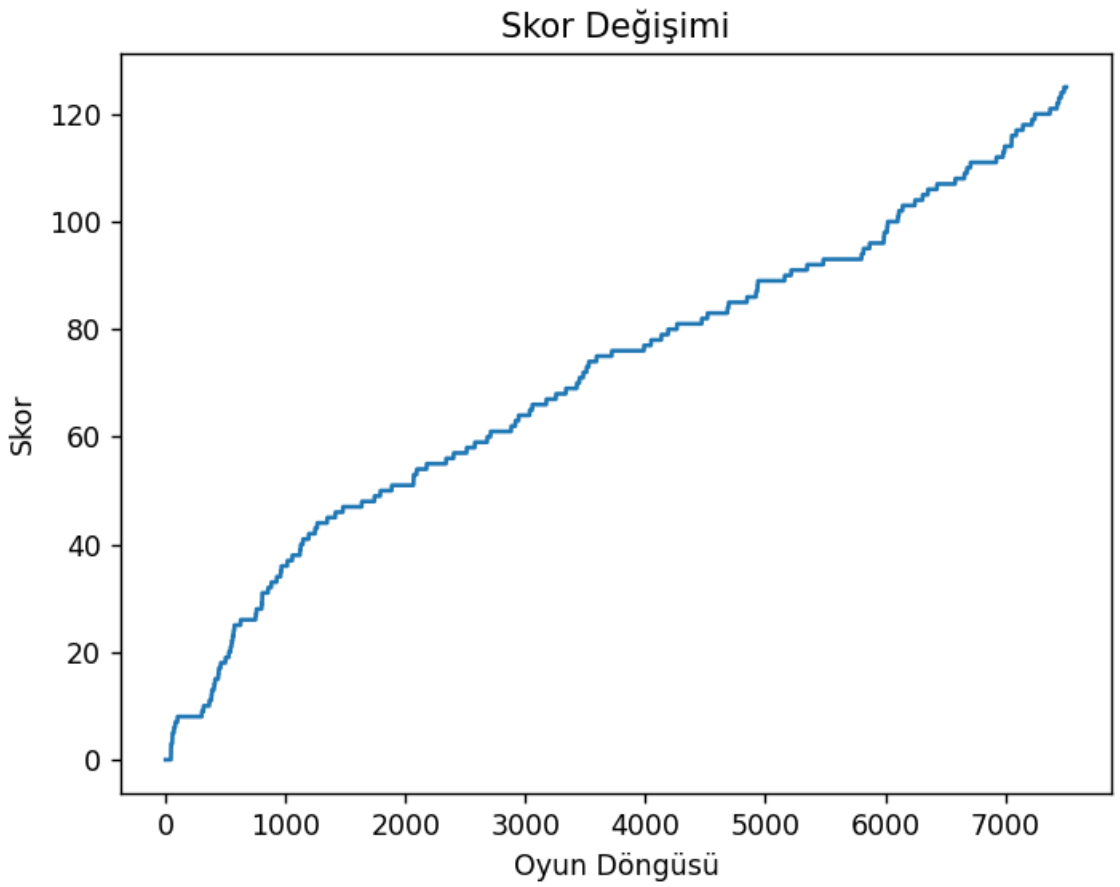
```
#Find the balloon closest to the mouse cursor
min_distance = float('inf')
closest_balloon = None
mouse_x, mouse_y = pygame.mouse.get_pos()

for balloon in balloons:
    distance = math.sqrt((balloon.x - mouse_x)**2 + (balloon.y - mouse_y)**2)
    if distance < min_distance:
        min_distance = distance
        closest_balloon = balloon

if closest_balloon is not None:
    if min_distance <= 200:  # Adjust the threshold as per your preference
        score += 1
    else:
        score -= 1
```

Buradaysa en yakındaki balonla arasında 200'den daha az mesafe varsa **score** değişkenini 1 artırarak ödüllendiriyoruz ve eğer yakınında bir balon yoksa 1 azaltarak cezalandırıyoruz.

Buradaki kontrol edilen mesafe değiştirilebilir.

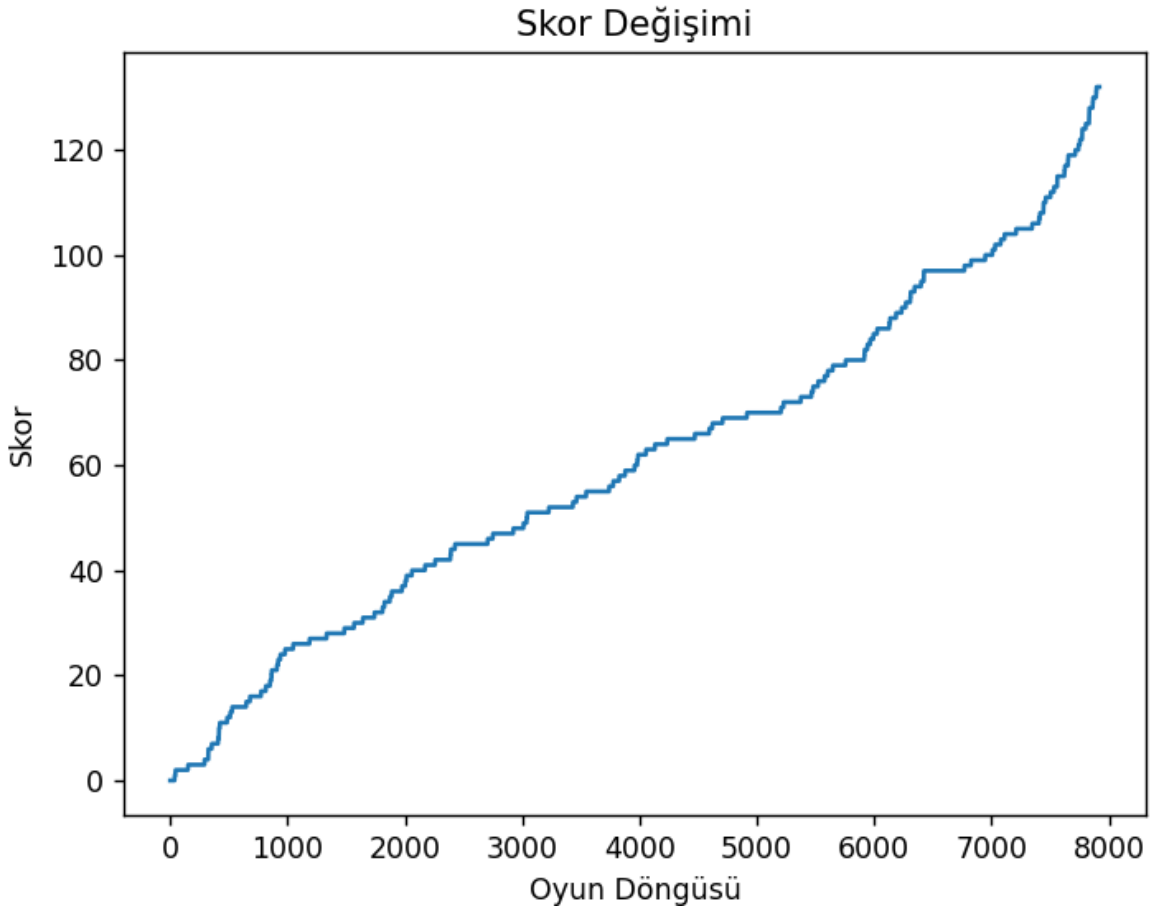
AJANIMIZI TEST EDELİM

İlk olarak 15 balon içeren ve en yakındaki balon olarak 200 birim mesafeye bakan bir kodu çalıştıralım. Yani ajanımız 200 birimden daha fazla mesafeyi göremeyecek.



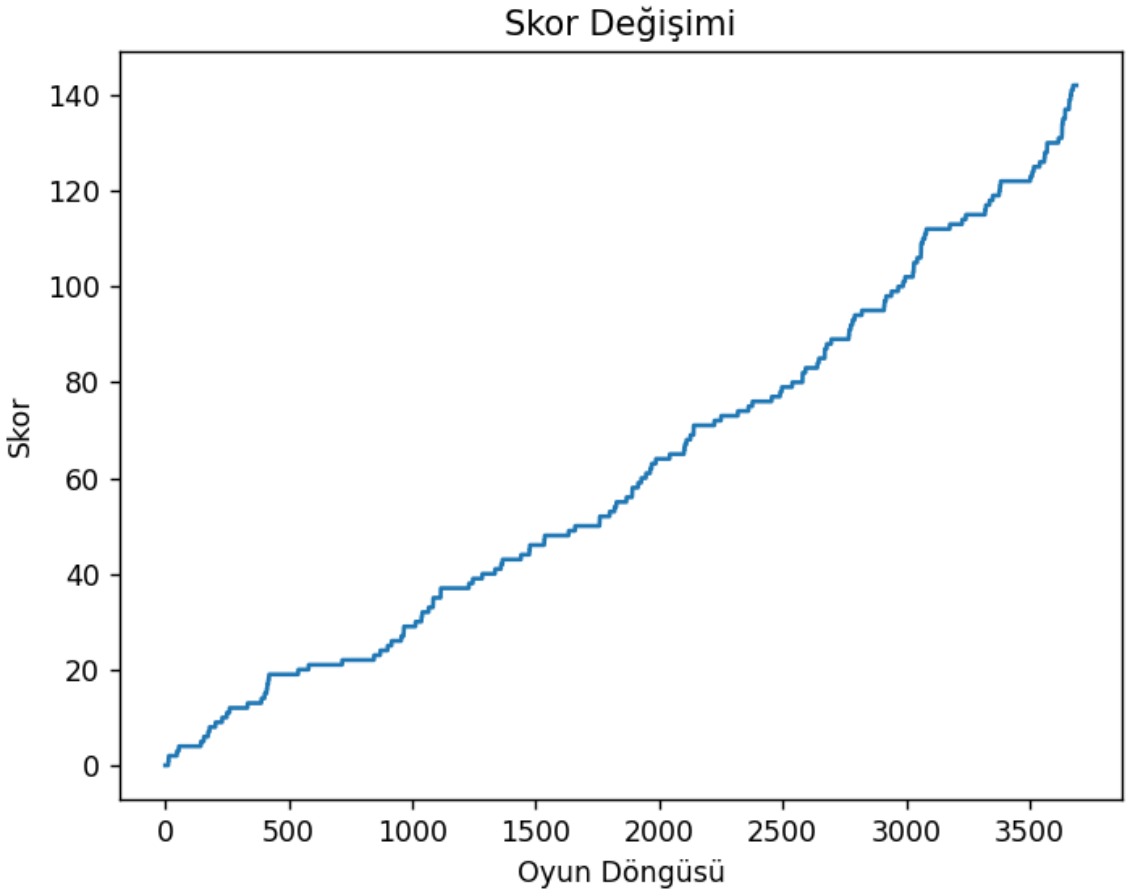
Soldaki değer patlatılan balon sayısını gösteriyor. En başta rastgele şekilde çalışıyor ve sonrasında hem iyi eylemleri öğreniyor hem de epsilon değeri zamanla azaldığı için bu öğrendiklerini daha çok eyleme döküyor ve bunun getirisi olarak sonlarda yükseliş artmış.

Şimdi de 15 balonlu ve 250 mesafeli bir örnek deneyelim. Bu sefer ajanın görüş mesafesi daha fazla.

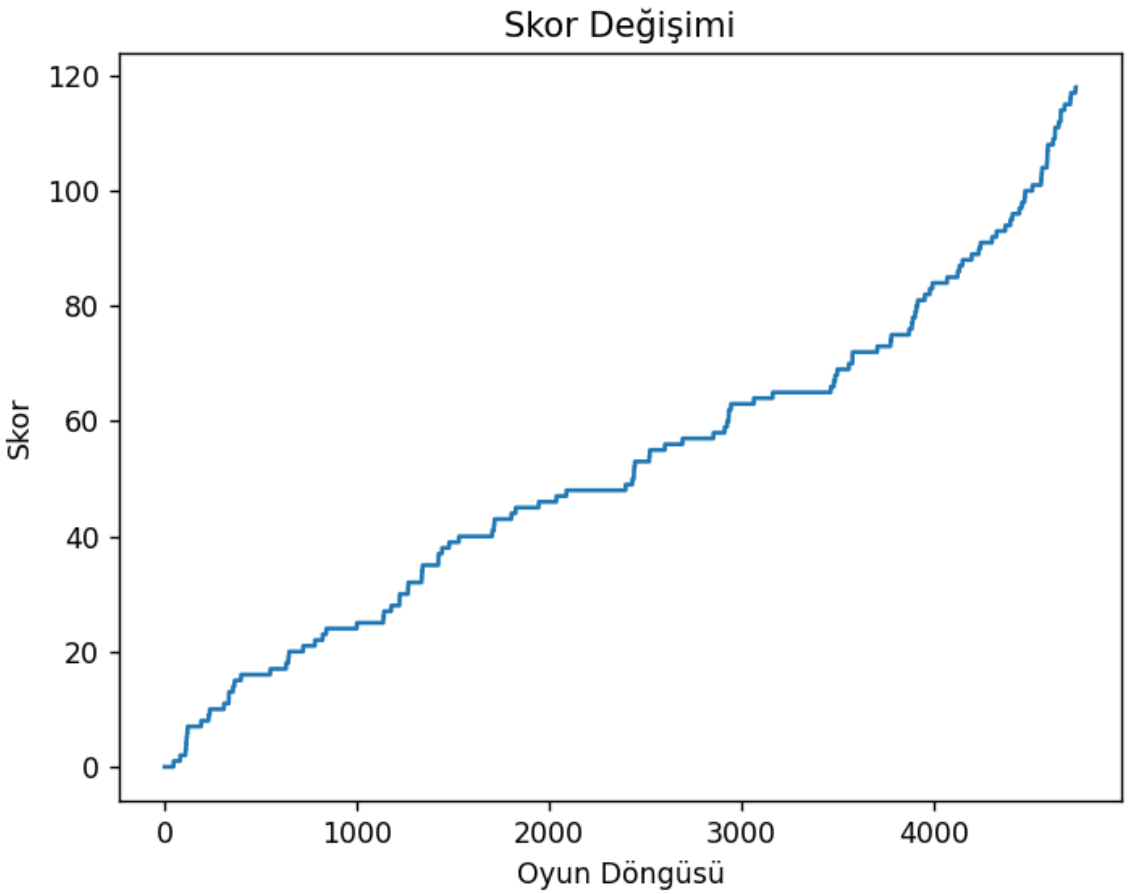


Benzer bir sonuç var. Alan çok da büyük olmadığı için 250 mesafe çok da etkili olmamış olabilir

Bu sefer görüş mesafesini 100 yapalım ama balon sayısını 20 yapalım. Bu sayede daha küçük bir alanı görse de alan başına düşen balon sayısı artar.



Ve son olarak farklı bir şey deneyelim. Yeniden 15 balon ve 200 görüş mesafesi yapalım ama bu sefer epsilonun azalma hızını azaltalım. Yani artık yeni deneyimlere daha açık ve tecrübelere kapalı bir ajan ile karşı karşıya olacağız.



Bunu sağlamak için **epsilon_decay** değişkenini 0.995 değerinden 0.999 değerine getirdik. Daha hızlı yükselen bir sonuç görebiliyoruz böylece.