MonsterKong - Agent - reinforcement learning

Alexandru Lucian - IIIA4
Onesim Robert - IIIA4

1. Etape:

- **Input: State-ul** - GameScreenRGB (getScreenRGB) **vector de pixeli**
- **Preprocesarea imaginilor**
- Calcularea functiei de scor (**reward-ul**) bazat pe state-ul jocului
- Introducerea imaginilor in **reteaua neuronala** (Q-learning algorithm - maximizarea reward-ul)
- **Output:** cea mai buna mutare (sus/jos/stanga/dreapta/saritura)

2. Input

Am folosit PLE: A Reinforcement Learning Env (http://pygame-learning-environment.readthedocs.io/en/latest/) pentru a simula jocul si a primi ca input state-ul acestuia.

3. Preprocesarea Imaginilor

- Transformam imaginile color primite in alb-negru
- Redimensionam imaginile in 80x80 pixeli
- Grupam cate patru imagini pentru ca modelul sa poata deduce viteza de miscare bilelor de foc si a monstruletului.

4. Reteaua Neuronala

Am folosit **Keras** cu **TensorFlow** backend

- 1 input layer - 4x80x80 image
- 3 convolution layers (hidden)
- 1 hidden fully connected layer
- 1 output fully connected layer- 5 noduri, cate unul pentru fiecare actiune
  Functia de activare folosite ReLu (f(x) = max (0,x)) - x input of a neuron
  Distibutie normala cu deviatia standard 0.01
  Am folosit ADAM (Adaptive Moment Estimation) pentru optimizarea retelei

5. Flow-ul antrenamentului

- vom avea un numar de frame-uri (EXPLORE), perioada in care vom construi initial baza de antrenament (deoarece la inceput nu avem date)
- un numar de iteratii in care se agentul se va antrena (TRAIN)
- lista (replay memory) la care se adauga constant frame-uri (si in faza de explorare dar si in faza de training). Dimensiunea maxima este 20000 urmand ca pe parcurs (cand se umple) sa stergem cele mai vechi frame-uri. Din replay memory sunt alese batch-urile pentru training.
- epsilon (gradul de explorare) - parametru care va fi initial o valoare intre 0-1.
  La fiecare pas al iteratiei se alege o valoarea random intre 0-1:
  - daca valoarea e mai mica ca epsilon , se va alege o actiune random astfel oferindu-se sansa agentului de a explora state-uri noi.
  - daca valoarea este mare decat epsilon atunci modelul va prezice o actiune pe baza state-ului curent model.predict(state);
- cu actiunea respectiva generam reward-ul si formam tuplu (s, a, r, s') care se adauga la replace memory
- in faza de training se alege un batch din replace memory de dimensiune 20
  I. Q(s,a) = r + gamma * argmax Q(s', a') reward-ul imediat + o valoare maxima viitoare rezultata dintr-o stare viitoare
  II. GAMMA = variabila de discount intre 0 si 1. Cu cat reward-ul e mai indepartat (mai viitor) cu atat mai putin il luam in considerare. Daca e 0, ne bazam pe reward-ul imediat.
  III. targets = model.predict(old_state)
  IV. newQval = model.predict(new_state)
  V. target[moves] = reward + gamma * max(newQval).
  VI. model.train_on_batch (inputs, targets)

Referinte importante:

http://www.cdf.toronto.edu/~csc2542h/fall/material/csc2542f16_dqn.pdf

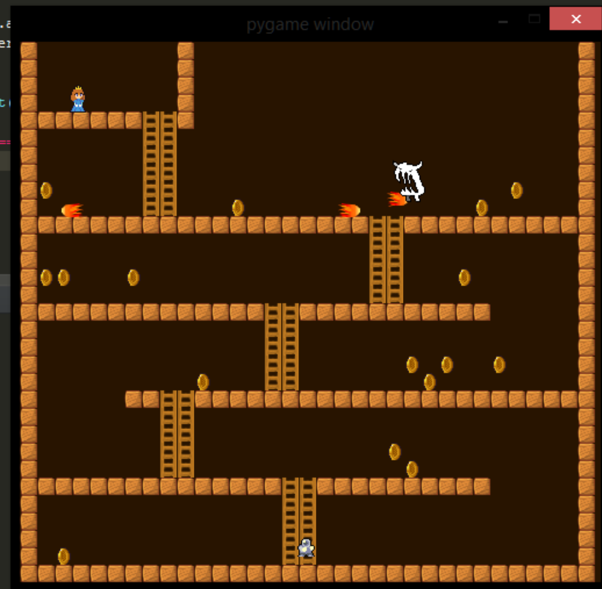
http://files.davidqiu.com/research/nature14236.pdf

http://sebastianruder.com/optimizing-gradient-descent/

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

```
FRAME 60062  EPSILON 0.14313899999994315  ACTION JUMP  REWARD 1.284722222222222
FRAME 60063  EPSILON 0.14313799999994314  ACTION JUMP  REWARD -6.215277777777778
FRAME 60064  EPSILON 0.14313699999994314  ACTION JUMP  REWARD -1.2486111111111111
FRAME 60065  EPSILON 0.14313599999994314  ACTION JUMP  REWARD 1.284722222222222
FRAME 60066  EPSILON 0.14313499999994314  ACTION JUMP  REWARD 1.284722222222222
FRAME 60067  EPSILON 0.14313399999994314  ACTION JUMP  REWARD 1.284722222222222
FRAME 60068  EPSILON 0.14313299999994314  ACTION JUMP  REWARD 1.284722222222222
FRAME 60069  EPSILON 0.14313199999994314  ACTION LEFT  REWARD 1.284722222222222
FRAME 60070  EPSILON 0.14313099999994314  ACTION JUMP  REWARD 1.284722222222222
FRAME 60071  EPSILON 0.14312999999994314  ACTION JUMP  REWARD 1.284722222222222
FRAME 60072  EPSILON 0.14312899999994314  ACTION JUMP  REWARD 1.284722222222222
FRAME 60073  EPSILON 0.14312799999994313  ACTION JUMP  REWARD 1.284722222222222
FRAME 60074  EPSILON 0.14312699999994313  ACTION LEFT  REWARD 1.284722222222222
FRAME 60075  EPSILON 0.14312599999994313  ACTION DOWN  REWARD 1.3541666666666665
FRAME 60076  EPSILON 0.14312499999994313  ACTION LEFT  REWARD 1.284722222222222
FRAME 60077  EPSILON 0.14312399999994313  ACTION DOWN  REWARD 1.3541666666666665
FRAME 60078  EPSILON 0.14312299999994313  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60079  EPSILON 0.14312199999994313  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60080  EPSILON 0.14312099999994313  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60081  EPSILON 0.14311999999994313  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60082  EPSILON 0.14311899999994313  ACTION JUMP  REWARD -6.145833333333334
FRAME 60083  EPSILON 0.14311799999994312  ACTION JUMP  REWARD -1.1791666666666667
FRAME 60084  EPSILON 0.14311699999994312  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60085  EPSILON 0.14311599999994312  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60086  EPSILON 0.14311499999994312  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60087  EPSILON 0.14311399999994312  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60088  EPSILON 0.14311299999994312  ACTION NOOP  REWARD 1.3541666666666665
FRAME 60089  EPSILON 0.14311199999994312  ACTION JUMP  REWARD 1.3541666666666665
FRAME 60090  EPSILON 0.14311099999994312  ACTION LEFT  REWARD 1.284722222222222
FRAME 60091  EPSILON 0.14310999999994312  ACTION JUMP  REWARD 1.284722222222222
FRAME 60092  EPSILON 0.14310899999994312  ACTION JUMP  REWARD 1.284722222222222
FRAME 60093  EPSILON 0.14310799999994311  ACTION LEFT  REWARD 1.284722222222222
FRAME 60094  EPSILON 0.1431069999999431  ACTION JUMP  REWARD 1.284722222222222
FRAME 60095  EPSILON 0.1431059999999431  ACTION DOWN  REWARD 1.3541666666666665
FRAME 60096  EPSILON 0.1431049999999431  ACTION LEFT  REWARD 1.284722222222222
FRAME 60097  EPSILON 0.1431039999999431  ACTION JUMP  REWARD 1.284722222222222
FRAME 60098  EPSILON 0.1431029999999431  ACTION JUMP  REWARD 1.284722222222222
FRAME 60099  EPSILON 0.1431019999999431  ACTION JUMP  REWARD 1.284722222222222
FRAME 60100  EPSILON 0.1431009999999431  ACTION JUMP  REWARD 1.284722222222222
FRAME 60101  EPSILON 0.1430999999999431  ACTION JUMP  REWARD -6.215277777777778
FRAME 60102  EPSILON 0.1430989999999431  ACTION JUMP  REWARD -1.2486111111111111
FRAME 60103  EPSILON 0.1430979999999431  ACTION UP    REWARD -1.2486111111111111
FRAME 60104  EPSILON 0.1430969999999431  ACTION DOWN  REWARD -1.1791666666666667
```

```
231            player.startNetwork(999999999, player.EPSILON_FINAL)
232
233        elif sys.a
234            player
235
236        else:
237            print(
238
239    if __name__ ==
240        main()
```




```
FRAME 2377  EPSILON 0.0001  ACTION JUMP  REWARD 3.4333333333333336
FRAME 2378  EPSILON 0.0001  ACTION UP    REWARD 3.6
FRAME 2379  EPSILON 0.0001  ACTION UP    REWARD 3.7666666666666666
FRAME 2380  EPSILON 0.0001  ACTION RGHT  REWARD 3.6
FRAME 2381  EPSILON 0.0001  ACTION NOOP  REWARD 3.6
FRAME 2382  EPSILON 0.0001  ACTION NOOP  REWARD 3.6
FRAME 2383  EPSILON 0.0001  ACTION RGHT  REWARD 3.4333333333333336
FRAME 2384  EPSILON 0.0001  ACTION RGHT  REWARD 3.2666666666666666
FRAME 2385  EPSILON 0.0001  ACTION JUMP  REWARD 3.2666666666666666
FRAME 2386  EPSILON 0.0001  ACTION RGHT  REWARD 3.1
FRAME 2387  EPSILON 0.0001  ACTION RGHT  REWARD 2.9333333333333336
FRAME 2388  EPSILON 0.0001  ACTION JUMP  REWARD 2.9333333333333336
FRAME 2389  EPSILON 0.0001  ACTION DOWN  REWARD 2.9333333333333336
FRAME 2390  EPSILON 0.0001  ACTION LEFT  REWARD 2.9333333333333336
FRAME 2391  EPSILON 0.0001  ACTION UP    REWARD 3.1
FRAME 2392  EPSILON 0.0001  ACTION DOWN  REWARD 3.1
FRAME 2393  EPSILON 0.0001  ACTION LEFT  REWARD 3.1
FRAME 2394  EPSILON 0.0001  ACTION JUMP  REWARD 3.1
```

```python
COIN_WEIGHT_Y = 5.
COIN_VALUE = 3.0

ACTIONS = 5                 # valid moves
EPSILON_INITIAL = 0.2       # starting value of epsilon
EPSILON_FINAL = 0.0001      # final value of epsilon


def __init__(self):
    self.buildModel()
    self.game = MonsterKong()
    self.p = PLE(self.game, fps=30, display_screen=True)


def buildModel(self):
    # build model using Keras
    self.model = Sequential()
    # input layer 4x80x80 image and 3 convolution hidden layers with activation function "ReLu" f(x
    self.model.add(Convolution2D(32, 8, 8, subsample=(4, 4), init=lambda shape, name: normal(shape,
    self.model.add(Activation('relu'))
    self.model.add(Convolution2D(64, 4, 4, subsample=(2, 2), init=lambda shape, name: normal(shape,
    self.model.add(Activation('relu'))
    self.model.add(Convolution2D(64, 3, 3, subsample=(1, 1), init=lambda shape, name: normal(shape,
    self.model.add(Activation('relu'))
    self.model.add(Flatten())
    # last hidden fully connected layer
    self.model.add(Dense(512, init=lambda shape, name: normal(shape, scale=0.01, name=name)))
    self.model.add(Activation('relu'))
    # output layer - 1 neuron for each valid move (5)
    self.model.add(Dense(5, init=lambda shape, name: normal(shape, scale=0.01, name=name)))

    # using ADAM (Adaptive Moment Estimation)
    self.model.compile(loss='mse', optimizer=Adam(lr=1e-6))

def saveModel(self):
```