Engineering Design Document
Dangerous Dave (qDeerein)

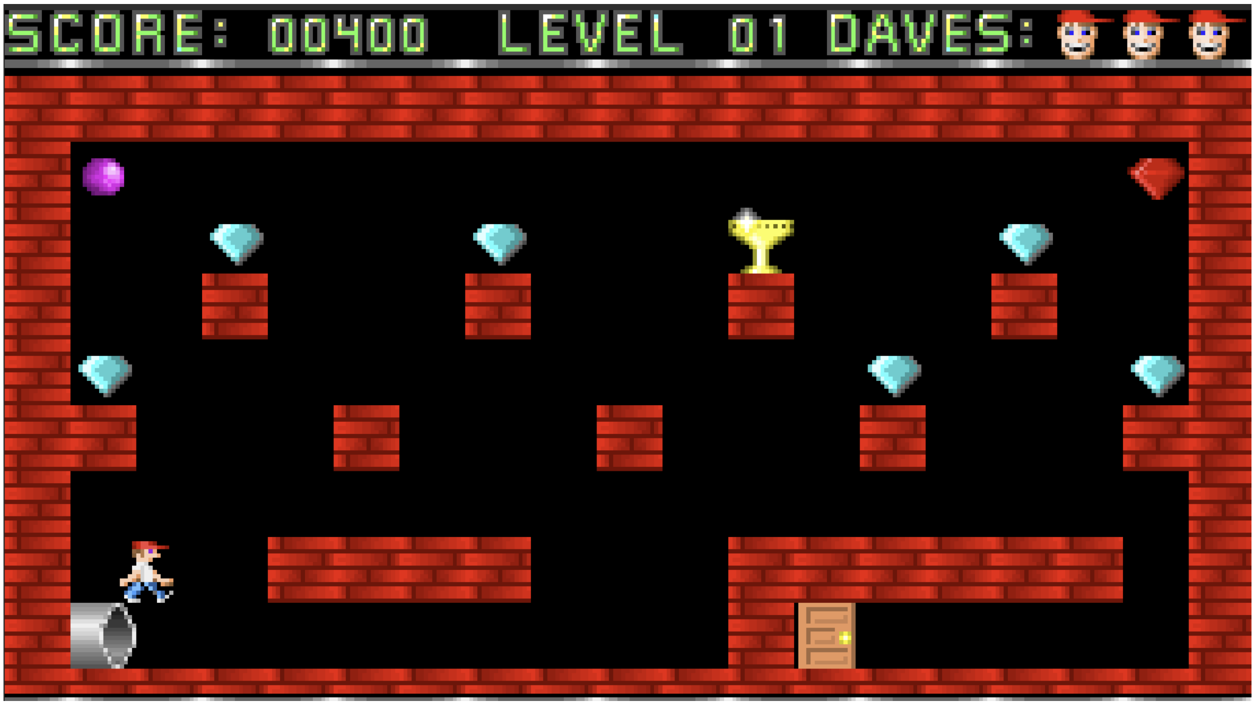
Akash Patel (akashkum@usc.edu)
Harima Patel (harimapa@usc.edu)
Jainam Divyesh Shah (jainamdi@usc.edu)
Mansi P Shah (mansipra@usc.edu)
Prabhat Mishra (prabhatm@usc.edu)
Yash Chanchad (chanchad@usc.edu)

GOAL OF THE PROJECT

The goal of the project is to train an agent that plays the ‘Dangerous Dave’ game. The objective is to prevent any human intervention and make the agent learn the optimal moves. This can be achieved by utilizing various reinforcement learning and deep learning techniques. There are various levels/stages in this game but the scope of this project is to utilize the 1st stage of the game for training and testing purposes.

OVERVIEW

Dangerous Dave is a 1988 single-player computer game developed by John Romero for the AppleII and DOS. The objective of the game is to explore the deserted pirate's hideout where Dave's rival, Clyde Cooper, hid ten skateboarding trophies. Dave has to collect gold cups to move on to the next levels. There are around 10 levels. On his quest, Dave will encounter various monsters, traps, and other perils, as well as treasure, useful items, and weapons.



GAME ENVIRONMENT

List of Moves:

Keys	**Action**
Left Arrow	Move Left
Right Arrow	Move Right
Up Arrow	Jump
Up Arrow + Left Arrow	Jump Backwards
Up Arrow + Right Arrow	Jump Forward

List of Rewards (Collecting them helps improve score)
Purple Ball - 50 points
Gem - 100 points
Red Gem - 150 points
Cup - 1000, Way to open the door for next level
List of Hurdles (Avoiding them helps improve score)
FearsomeFire, WeirdWeeds, WickedWater will kill Dave.

MOTIVATION

A large portion of the world's population is attracted by the gaming sector. With a growing consumer base, the trend of applying Machine Learning concepts to train an Artificial Intelligence agent is having ample opportunities to innovate and provide compelling content with social and cultural context.

Using Machine Learning concepts we wanted to imitate the human learning process of adapting to improve performance to understand how the human brain works.

Instead of using supervised learning, which requires prior knowledge of what action the agent should take given a state of the environment, Reinforcement Learning is used where the agent can see the results of its actions, evaluate its performance, and learn from mistakes. By using Deep Q-Network, which uses a neural network having an architecture similar to the human brain, the agent is allowed to explore an unstructured environment and acquire knowledge with time by making it possible to imitate human behavior. Standard Q-Learning would fail as the state space increases, so Deep Q-Network is used to train an agent that works where the number of states overwhelms the capacity of contemporary computers.

The target is to create an agent that could work on all different types of similar games as the agent does not depend on the environment specific to this game.

RELATED WORK

Deepmind Technologies created a deep learning model to train an AI agent to play Atari games with no adjustment of architecture or learning algorithm. This agent outperformed human agents. It used reinforcement learning to learn policies from high-dimensional sensory input like vision and speech. The model used was a convolutional neural network whose input is raw pixels and output is a value function estimating future rewards.

The best example of the use of reinforcement learning in the past is TD-gammon, which is a model-free algorithm that uses Q-learning and approximates the value function using a multi-layer perceptron with one hidden layer.

Deepmind Technologies created a program that combined advanced search and deep neural networks to create an agent called AlphaGo to play the Go board game. One neural network selects the next move, this is called the policy network or actor. The other neural network predicts the quality of the move, this is called value network or critic. This reinforcement learning method is called the Action-Critic method. AlphaGo went on to defeat Go world champions.

Deep Mind and the University of Montreal present asynchronous variants of standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training.

To date, there is no commendable work on using Reinforcement learning and Deep Neural network for training Dangerous Dave game agents.

BACKGROUND RESEARCH TASKS

Our research towards completing the project:

- Read different papers containing the implementation of various Machine Learning techniques for training agents.
- Referred multiple video tutorials and articles along with lecture slides.
- For this DOS-based game, we didn't have access to the source code, due to which we researched on:
    - Simulating multiple key presses simultaneously in Python using existing libraries win32con and win32api.

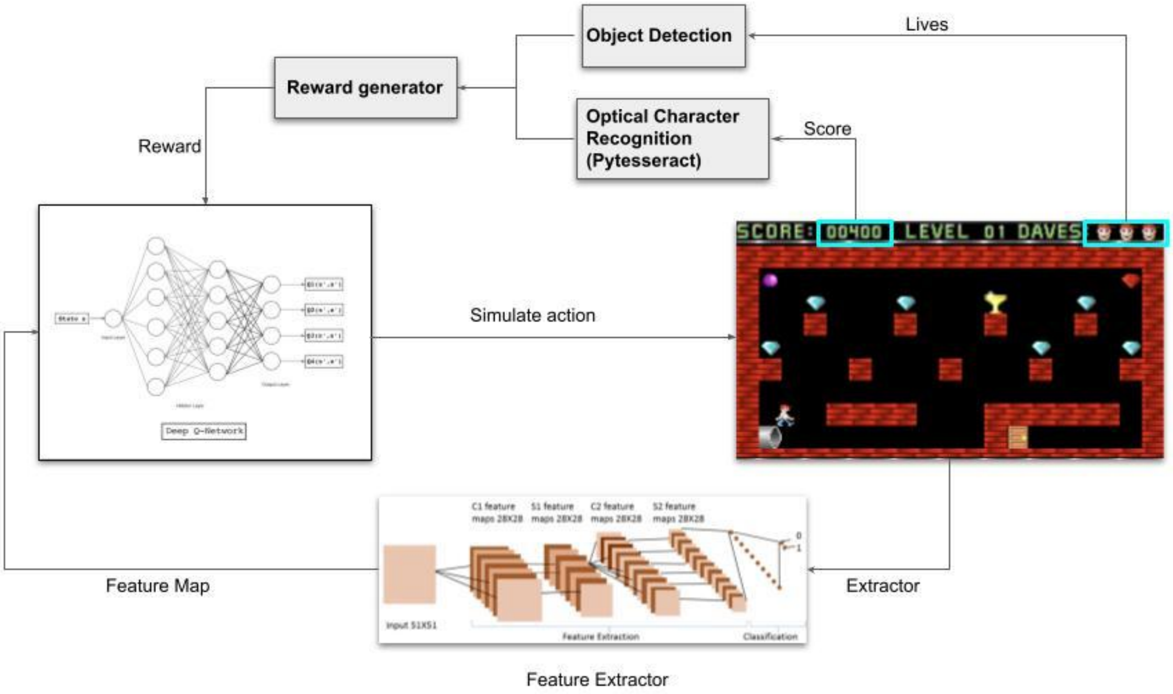
- Capturing a portion of the screen using the cv2 library and feeding it to CNN to extract features. The purpose of feeding a portion of the screen, instead of the whole screen is to feed the image of the ideal size that would translate necessary details and improve the performance of CNN.
- Fetching data from images using pytesseract for score and number of lives.

- Explored Keras, TensorFlow, and PyTorch to implement Neural Network and Deep Learning.
- Researched the Deep Q network, Q learning algorithm, Actor-Critic method, and its implementation.
- Studied about mobile nets in Keras and extracted features from an image using it.
- Learned about OpenAI gym and its various environments.

PROJECT FRAMEWORK AND MODEL ARCHITECTURE

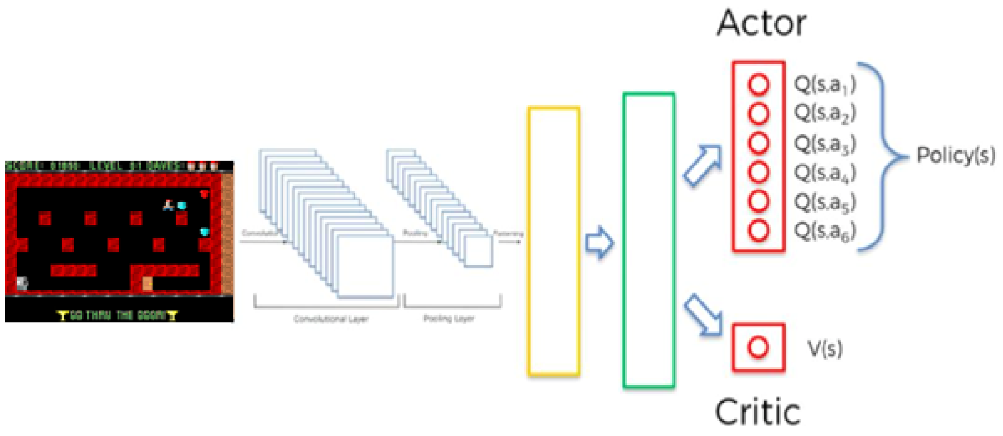
1. Using DQN

Deep reinforcement learning is based on value function. Earlier approaches of reinforcement learning approximate value function based on rewards but in this approach it is possible to approximate functions through deep neural networks. Following architecture describes flow using DQN.

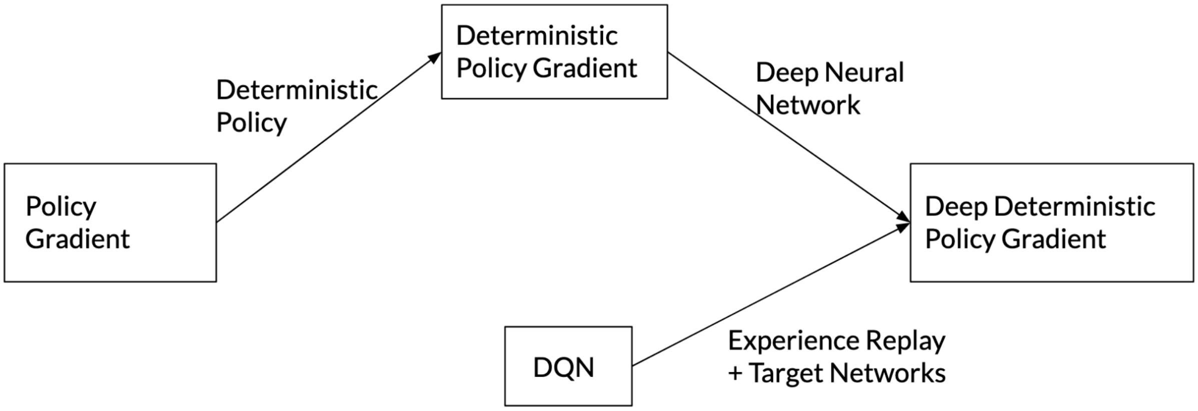


2. Using DDPG

The following is an architecture that augments a new neural network with our previously developed neural network. This new neural network acts as an actor that selects the best action believed by the agent in a particular state. This action is evaluated using the critic network. Essentially, the actor network is estimating the optimal target policy and the critic network is estimating the action-value function corresponding to target policy.



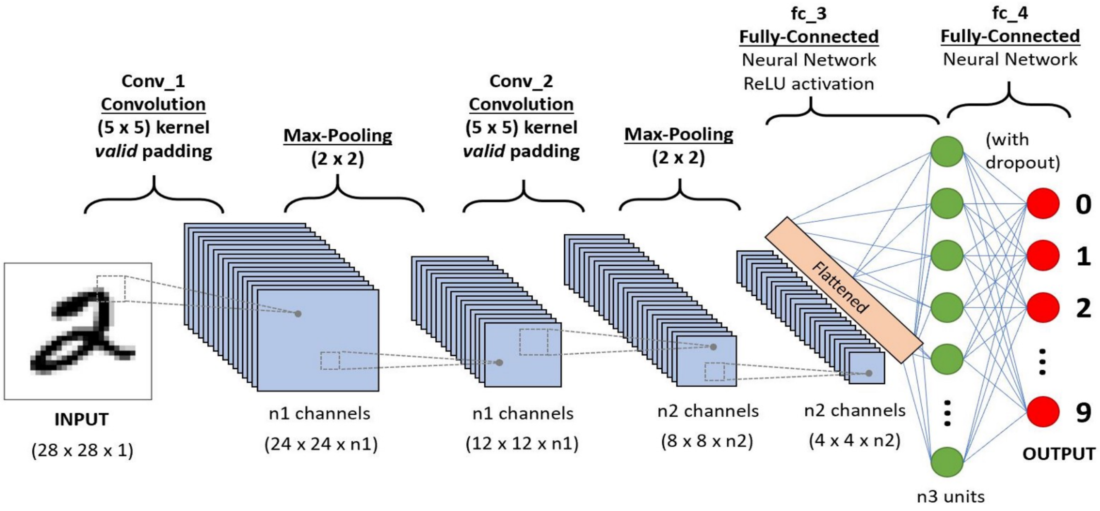
The following is a flow of a variant of Actor-Critic model called Deep Deterministic Policy Gradient. The policy gradient selects a deterministic policy by evaluating states and action from the output of the previous iteration of the algorithm. The idea is to update the policy in the direction of the gradient of critic network.


Deterministic Policy Gradient
Deterministic Policy
Deep Neural Network
Policy Gradient
Deep Deterministic Policy Gradient
DQN
Experience Replay + Target Networks

METHODOLOGY

1. Using DQN

1) Initially, the DOS game Dangerous Dave will be called using Python code.
2) Once the game starts in full screen, ie. in the 1366 x 768 window, the Python program captures the game environment using ImageGrab library.
3) The image grabbed will be fed to **Feature Extractor,** a Convolutional Neural Network, to generate the Feature map which is used to represent the state of the game environment that would help in recognizing the location of the Agent (Dave), location of gems, the location of obstacles. This is done using the TensorFlow python library. TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.



A **Convolutional Neural Network** can successfully capture Spatial and Temporal dependencies in the grabbed image and reduce images that can be processed easily without losing any features that are critical to defining the game environment. The kernel extracts high-level features by performing convolution operations. The average pooling layer reduces the spatial size of convolved features to reduce the processing power required.

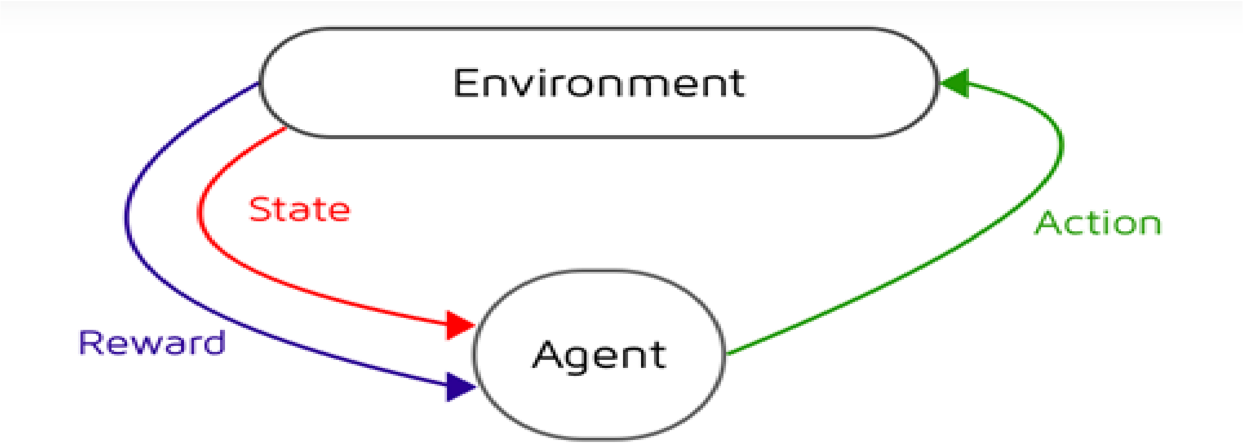
4) The action is selected using the **Epsilon-Greedy approach**, that is it is either the action predicted by Deep-Q-Network or random action. With time epsilon is adjusted so that the probability of exploration (choosing the random move) decreases and exploitation (choosing move with maximum q-values) increases as the training progresses.
5) The action can be one of the following: <jump right, jump left, move left, move right>. Deep-Q-Network is a neural network that returns different pairs of actions and q-values for a particular state (game environment) from an infinite state space. It is a type of

Q-Learning algorithm which is based on **Markov's Decision Process**. The initial model to predict possible actions and respective q-values is created using the **Sequential model by Keras**, adding dense layers with relu activation function, and using SGD for optimization. **Stochastic Gradient Descent** is an optimization algorithm that estimates the error gradient for the current state of the model using examples from the training dataset, then updates the weights of the model using the back-propagation of errors algorithm.

6) The action is then performed and observed in the actual Dangerous Dave DOS game environment and the resulting score is read from the top-left corner of the screen by converting color image to black and white image, resizing it, and converting image to text using pytesseract.
7) Based on the score, we assign rewards to that particular action and save the initial state, action, final state, reward for future training. We can prevent q-values from oscillating or diverging by training on random minibatch from experience, instead of using the latest experience. This method is called **Experience Replay**. It helps in breaking harmful correlations, learn some tuples multiple times and recall rare occurrences. On every action played, we check the number of lives remaining and also check whether the trophy has been collected by the agent or not.
8) The complete experience is stored and fed back to Deep-Q-network to adjust its policy and learn different actions using **Reinforcement Learning**. The network weights are updated using the **Bellman equation**.

$$Q(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a\, Q(S_{t+1}, a))$$

9) The training occurs for around 100 games with 40, 60 and 80 epochs(moves) for each game.

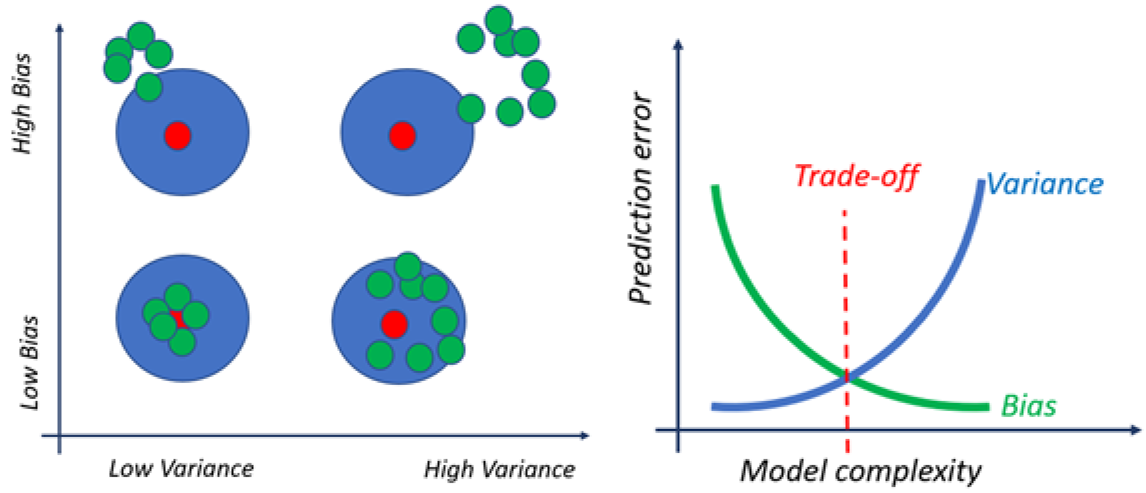


2. Using Actor-Critic

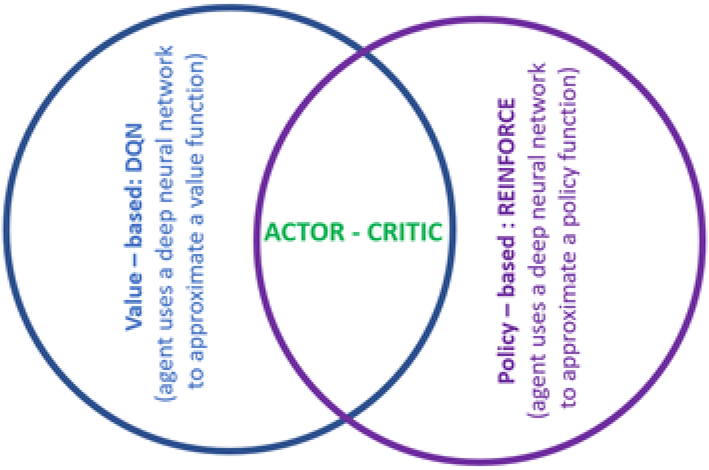
The problem after applying DQN was that the number of moves after learning the environment was significantly high. Also, the agent was not able to determine the direction of the door after learning the environment. The idea we get after reading Continuous control with deep reinforcement learning paper is that we can improve Q values by seeing more into future rewards. The future rewards include the reward of the door too; so we tried to add another neural network in DQN to create an actor-critic model. We were confident this approach would produce good results.

In **Actor-Critic** the idea is to split the model in two: one for computing an action based on a state and another one to produce the Q values of the action. The actor takes as input the state and outputs the best action. The critic evaluates the action by computing the value function. This works better than both value iteration and policy iteration, by converging faster.

Deep Deterministic Policy Gradient is a solution using Deep Q Learning in continuous action space. In the previous approach, we learn the optimal action-value from sampled state transitions(Experience replay) and output a discrete action-value function that selects the best possible action. This approach uses reinforcement learning that uses a deep neural network to estimate optimal policy directly from a given state and outputs a probability distribution. The previous method (DQN) tends to have high variance and low bias. The estimate is compounded resulting in wrong q values and hence the agent is not able to detect the last goal (door) which is required to advance to the next level. Policy-based methods have high variance and low bias as they use the Monte Carlo estimate to calculate discounted reward observing out of the whole sequence.



An actor-critic model tries to use both methods and give better Q values.


Value – based: DQN
(agent uses a deep neural network
to approximate a value function)
ACTOR - CRITIC
Policy – based : REINFORCE
(agent uses a deep neural network
to approximate a policy function)

The DDPG algorithm exploits the experience replay and target network of the Deep Q neural network. The amount of work after implementing DQN reduces significantly because we just need to implement an actor neural network. Given the duration of the project, this was the best solution to implement which significantly improves the performance and objective of the agent. DDPG uses random processes to generate exploratory actions. These random walk methods are used to model noise. Using a random process the agent interacts more with the environment. When constructing the target value, use the target actor to select the action, and use the target Q value to evaluate its target, which makes learning stable. The target will be updated slowly because of a delay factor tau which is a configurable hyperparameter.

The steps are updated using the following algorithm in the actor-critic environment.

1. Create a runner object that handles the different environment steps.
2. When the AI runner agent takes a step, each environment is updated with a corresponding action/step.
3. The batch is generated from different experiences from the environment.
4. Made a baseline model for DDPG and used batch experience for the training model.
5. The resulting batch of experiences is passed for gradient calculation.
6. The weights are updated and training is driven configuring various hyperparameters. Below is the algorithm of the process.

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

            $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$

        Update the target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$

    end for
end for
```

CHALLENGES

1. Environment setup

Installing Tesseract-OCR, Tensorflow, Keras, and Pywin32 libraries with correct and compatible versions was challenging After some research and development, the current versions we are using for the development are as follows:

TensorFlow (2.4.0) - an open-source machine learning platform
Tesseract-OCR (5.00) - an open-source optical character recognition engine
Python (3.6.8) - an interpreted, high-level, and general-purpose programming language
Pywin32 (300) - a windows extension to simulate the random moves on the actual DOS game

2. Running DOS game from Python code

Dangerous Dave is a DOS-based game and we need to start and end games from a python program in order to train and test our agent. In the process of training and simulating random moves on the actual DOS game, we had to write python scripts to start the DOS game and simulate the moves. As we couldn't directly invoke the DOS game environment from the Python project, we extracted the required DOS executable files for the game and copied them to our project directory.

3. Simulating random move

To train our agent, we need to make the player move in the DOS game environment by simulating key press through python code. Our game has 5 types of moves, and each move requires a single key or combination of keys to be pressed for a certain duration. Determining the duration of each keypress was a challenge. For example, to simulate the "jump left" action, we need to press the up arrow key for 0.3 seconds and after releasing the up arrow key, we need to press the left arrow key for 0.5 seconds before we release it.

Below is the list of moves and their respective key with the time duration of key press.

Actions	Keys to be pressed	Time(Seconds)
Left Move	Left Arrow	0.3
Right Move	Right Arrow	0.3
Jump up	Up Arrow	0.3

Left Up Jump	Up Arrow	0.3
	Left Arrow	0.5
Right Up Jump	Up Arrow	0.3
	Right Arrow	0.5

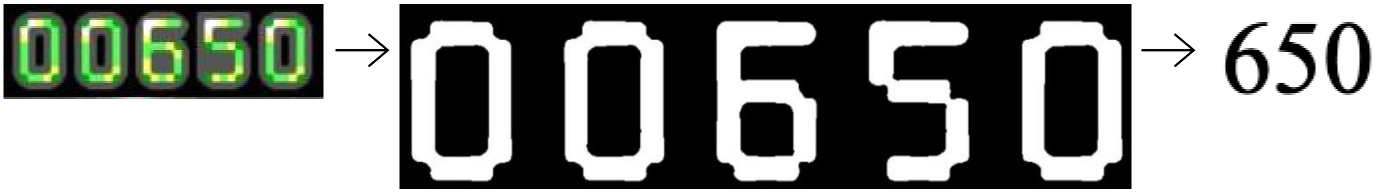
4. Fetching the score and number of lives remaining from the game environment

We need to fetch three parameters from the game environment:-

1. Player’s score (available at the top-left corner)
2. Player’s number of lives remaining (available at the right-top corner)
3. Whether trophy has been collected or not (available at the bottom)

1. Player’s score (available at the top-left corner)

We extract the player's score from the captured screenshot by cropping the top-left corner and reading specific screen coordinates. As dangerous dave is a low-resolution game, the cropped image is also of low resolution, which makes it difficult to extract the player's score.
According to our plan of action, we used the Tesseract-OCR library to extract scores from images. We tune the image by either reducing all pixels to zero (for the ones whose value is less than 120) or increasing the pixel value to 255 (for the ones whose value is more than 120). As a result, we eliminated noise from the image which in turn helped Tesseract to identify the text from the image more accurately. Also, using a different Tesseract-OCR (version 5- the one which supported low-resolution) and resizing the image was helpful in extracting text from a blurred image. The flow diagram below explains the detailed work.

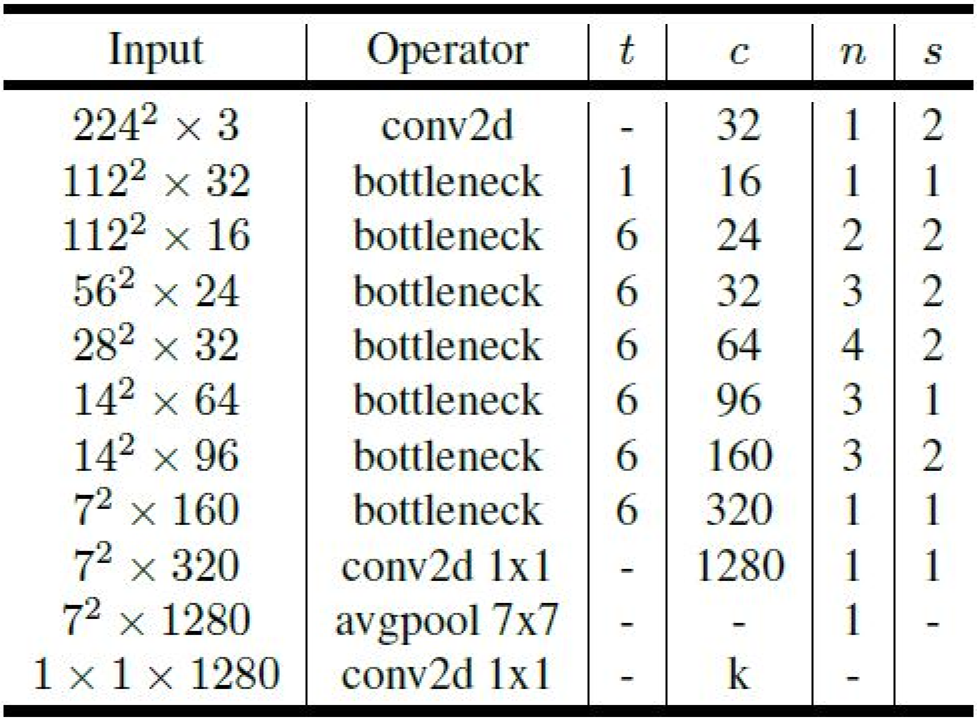


2. Player’s number of lives remaining (available at the right-top corner)

The challenging part in fetching player's number of lives was that there was no text to read and hence Tesseract OCR library was not useful. The cropped image depicting the number of available lives is divided into three subparts, each representing one life. Each one is compared with a blank black image and if they are similar, we do not increment the number of lives, as no face appears when the player has lost a life.

5. Improving CNN for feature extraction

As explained earlier, CNN uses convolutions to aggregate spatial information within the image. The neural network efficiency depends on model size as well as the number of floating-point operations in the network. These increase quadratically with respect to the kernel size. Usually, the kernel size is 3*3 but that size did not give satisfactory results in a 3 layer network. For example, the VGG-16 model with 3x3 conv leads to 15 million parameters. A lot of research has been done to reduce the size of spatial computations and there are a plethora of neural network architectures that help reduce the size of convolutions. However, there exist depth-wise separable convolutions that can be used instead of spatial convolutions which reduces the number of computations by 8 to 9 times. One such architecture is MobileNetV2. MobileNets uses 2 new global hyperparameters resolution multiplier, width multiplier. Using these parameters it is possible to increase the speed of learning and reduce hardware requirements due to low size by trading off accuracy. Since we ran our model on a laptop the accuracy of the move was not important because our environment is also static. Following is the architecture of MobileNetV2.



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

We used the conv2d operator to extract features from our game environment.

C - number of output channels

T - expansion factor

N - number of times the block is repeated
S - stride

6. Making the agent take a different move after reaching a dead end.

The agent was taking the same move when it reaches a dead end and was stuck in the same state because it selected the move with max q-value at a given state, so until the matrix difference between the current and next state after an action is not high enough, we select the next best action (action with next highest q value). The feature extractor takes a screenshot of the game environment as input and provides a matrix of the current state.

7. Solve the loop problem.

The DQN Agent is used to select the best move (the move with the highest q value). The problem was that at times the agent would keep repeating the same moves forever and stay in the same region. Eg. jump right, jump right, jump left, jump left, jump right, jump right, jump left, jump left.

We tried to detect the loop by remembering the timestamp when each move was taken and if the gap between the two same types of moves was the same every time, it was a loop. This could detect loops where all moves were unique like [jump right, left, jump left], but not where moves were repeating [jump right, jump right, jump left, jump left]. So we remember the last 8 moves and last 4 moves and if the last 4 moves were a part of the last 8 moves, the loop is detected. To break the loop we select the next best move. The problem with this method is that if some moves are repeating, for example [jump right, jump right, jump left, jump right, jump right, jump left], it would be detected as a loop when it is not a loop. We can detect loops by seeing if the sequence of positions of Dave has repetitions. We remember the state of the game environment that is output by the feature map and if the same sequence of states is repeated, it means there is a loop. This method does not detect loops if the gem is collected during the 1st iteration, as states would be different in the 1st and 2nd iteration. It would be able to detect the loop after the 3rd iteration.

8. Deciding which Actor-Critic Network to use

We also explored A2C, A3C, ACKTR, and PPO (baseline networks) , but according to research training should be done on environments generated by openAI Gym. So we decided to use DDPG because we had already worked on environment generation for the Critic network.

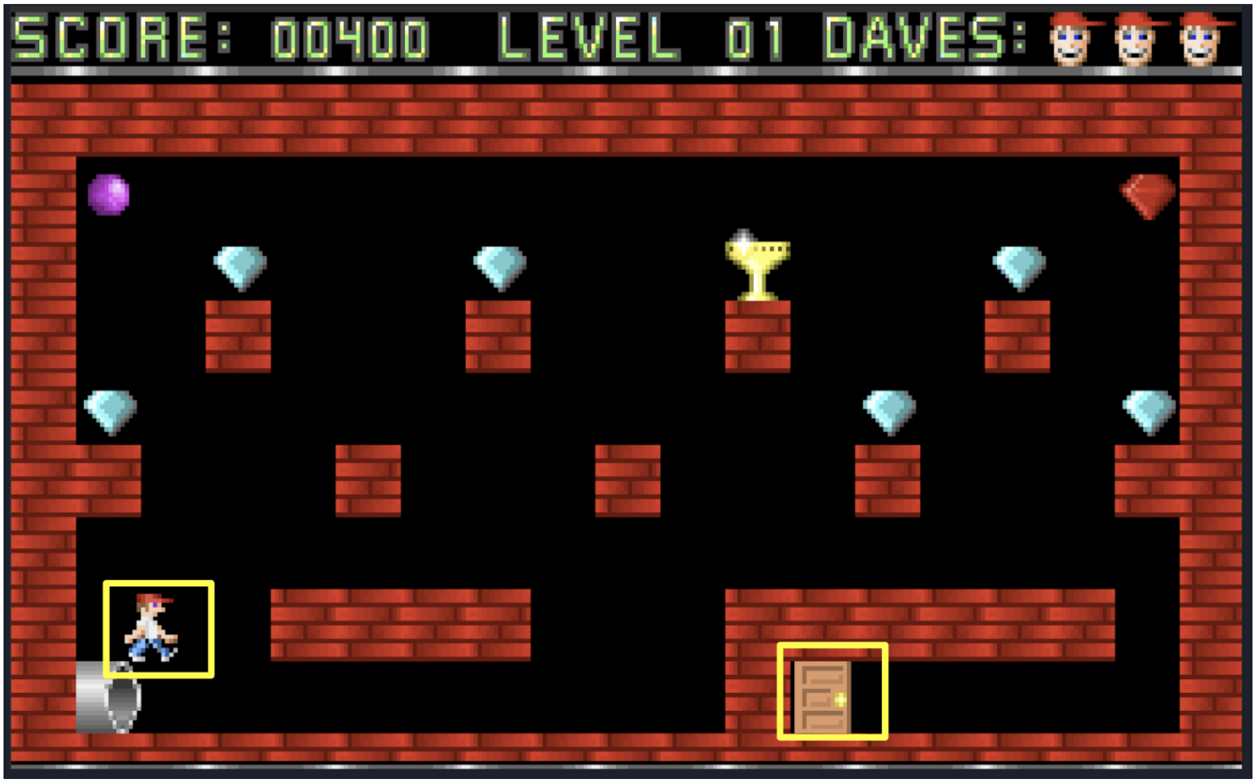
Parameter	Value in DQN	Value in Actor-Critic	Description

		(DDPG)	
Batch_Size	32	128	The number of experience tuples that are sampled from experience memory for experience replay.
No. of steps	40,60,80	40,60,80	The number of moves made by the agent in the game.
No. of epochs	618	322	The number of games played for training. Q-values would improve with time.
eps_start	1.0	1.0	The starting value of epsilon, for epsilon-greedy action selection, initially it is 1 so that all moves are selected randomly.
eps_decay	0.001	0.001	A factor to decrease epsilon so that we can move from exploration (making random moves) to exploitation (select moves predicted by DQN) as training progresses.
Rewards	Between -1 and 1	Between -1 and 1	If Dave has taken the trophy reward is 1. If Dave has gained a score of more than or equal to 50, he has picked some gem, so the reward is according to the gem score. If there is no significant change in the environment by a moving reward is -0.1. Otherwise reward is 0.01
Actor Neural Net Learning rate	NA	0.001	The learning rate of Actor Neural network.
Critic Neural Net Learning rate	0.001	0.001	The learning rate of Critic Neural Network.
Tau	NA	0.125	Polyak averaging factor : The target network is updated once per main network update by polyak averaging.

EXPERIMENTS

A variation of A* search for path detection to reach door

The drawback of DQN was that it decided the next best move only on the basis of Q-value returned from Deep Q-neural network. Because of that, players aimlessly wander in the game environment after collecting all the gems and trophies. So, our next goal was to detect the door and reach there in a minimum number of moves possible to reduce the number of steps required to finish the game. For that we explored path detection algorithms like A* to send the player towards the door after she collects all the gems and trophies. In this path detection algorithm, the next best move was decided on the basis of Q-value output from DQN and the heuristic function used in the path detection algorithm. The heuristic function that we used in A* algorithm was the Manhattan distance between the player's current position and the door. Here, the positions of player and door were determined using OpenCV.

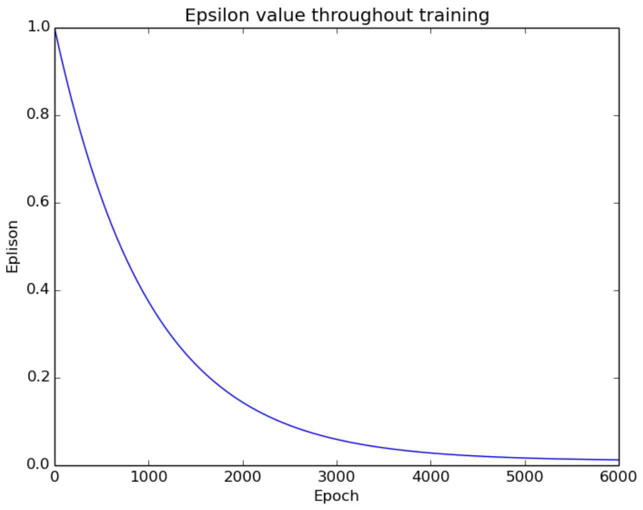


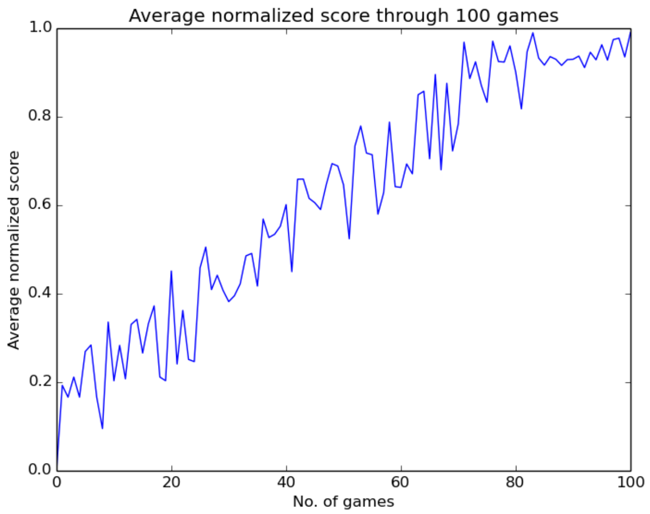
The Q-value guides the player to collect as many gems as possible and on the other hand, A*'s heuristic guides the player to move towards the door.

First, we detect the door's position and player's position using OpenCV and convert their position into 2D coordinates at each state. The moves which take the player closer to the door will be prioritized based on the heuristic function’s value which is essentially the Manhattan distance between the player’s current position and the door. So this implementation reduces the number of moves to reach the door after collecting all gems but it does not reduce the total number of steps of the entire game which is the drawback of this approach.

Methods	DQN	DQN with A* implementation	Actor-Critic
Average number of steps required to finish the game for 100 games	98	78	55

RESULTS AND STATISTICS

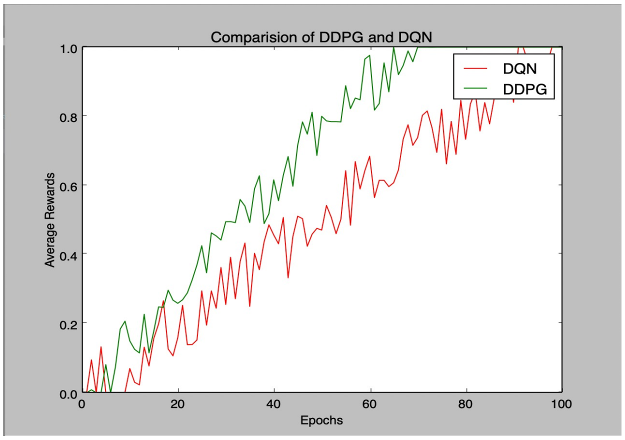


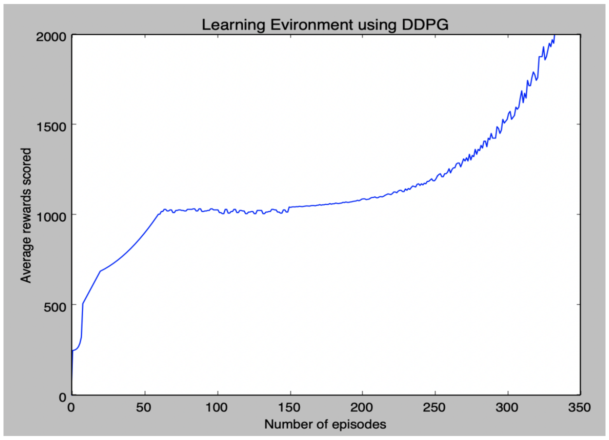


Initially the value of epsilon is 1. The high value of epsilon indicates high randomness in the player's behavior and the low value of epsilon indicates low randomness. With time, as the agent gets trained, the value of epsilon decreases, thus the probability of selecting moves generated by Deep Q-Network or Actor-Critic Network increases, so the agent makes better moves.

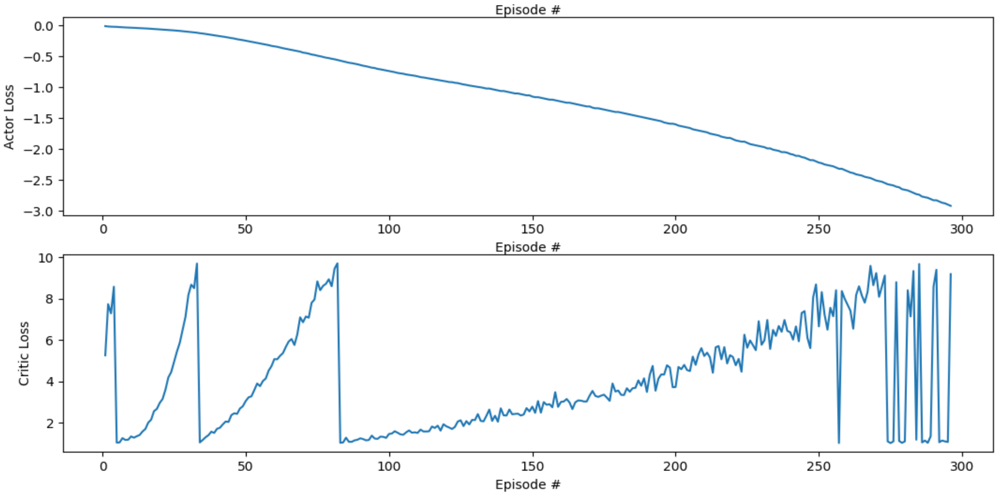
We allow the agent to play a fixed number of steps for 1000 games. We ran each game with 40, 60, and 80 steps and observed that we got best results in 60 steps.

The results of DQN are impressive indicating that the agent is able to evaluate action in a state and able to learn the environment but was not able to reach the door. After implementing Deep Deterministic Policy Gradient(DDPG) the number of moves in one game and number of episodes to learn the environment both reduced as expected. The reason for this result is better Q values.The agent learns the environment in 321 episodes in DDPG compared to 632 episodes in DQN. The agent reaches the door after collecting gems in an average of 60 steps in DDPG compared to 100 steps in DQN.





The training loss of the actor network shows similarity with the DQN neural network but critical network loss looks quite unusual. This anomaly could be possible because of change in experiences present in the replay buffer. Following graphs depict the loss observed during the entire training for both the networks. The Actor and Critic networks are running concurrently and hence results may change slightly if run on different infrastructure or computing configuration.



LIMITATIONS, FUTURE WORK, AND CONCLUSION

The environment in which the agent plays is static and the interaction only depends on the moves of the agent. Therefore, the approach suggested in this design might not work for a significantly dynamic environment that is not Atari-based like Starcraft, ViZDoom, Dota, etc, and adversarial-based games like chess. However, the design based on reinforcement learning combined with a deep neural network to learn Q values (DQN) has proved successful in learning the optimal policy for this game. The game is still far from playing better than a human player because the number of moves that are required to advance to a new level was never achieved even after training the agent for epochs ranging from minimum moves to a very large random move. Since this game does not require to have a time-bound move this parameter is not of great importance but can be important for other games.

Current work involves a design where the agent has no prior knowledge of the game and it learns to play it through image pixels, scores, and the number of lives the agent has in the game. This neural architecture can be tuned for this game and any other game by just configuring various hyperparameters. The experience replay algorithm and target network enhanced the experience of the agent and led to a stable learning network.

We tried actor-critic using DDPG to reduce our effort to solve problems of the game in a short duration but there are several implementations of this algorithm. We will try more algorithms such as Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Distributed Deterministic Policy Gradients (D4PG) and Truncated Natural Policy Gradient (TNPG) to see if we attain better performance for this game. We would also see if the same neural network model can be used for the level which involves obstacles. That implementation will only require to change the reward system and tuning of hyperparameters.

The current implementation of DDPG uses minibatch taken uniformly from the replay buffer. Alternatively, we can try using a prioritized replay buffer to compare the results. Also, we spent a great deal of time tuning hyperparameters but there may be other sets of values that make the agent solve the environment even faster.

Furthermore, we would like to see if this approach will work for other Atari games similar to dangerous dave where we need to avoid obstacles and collect rewards to move to the next level.

REFERENCES

1. https://www.giantbomb.com/dangerous-dave/3030-29215
2. https://en.wikipedia.org/wiki/Dangerous_Dave
3. https://www.freegameempire.com/games/Dangerous-Dave/Run-in-browser
4. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
5. Deep Q-Network (DQN)-II. Experience Replay and Target Networks | by Jordi TORRES.AI | Towards Data Science
6. https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a
7. https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
8. https://medium.com/analytics-vidhya/deep-q-network-with-convolutional-neural-networks-c761697897df
9. https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c#:~:text=The%20act%20of%20sampling%20a,better%20use%20of%20our%20experience
10. https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d
11. https://towardsdatascience.com/an-intuitive-explanation-of-policy-gradient-part-1-reinforce-aa4392cbfd3c
12. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
13. https://en.wikipedia.org/wiki/Convolutional_neural_network
14. https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
15. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/
16. https://www.youtube.com/watch?v=lvoHnicueoE&t=3077s
17. https://towardsdatascience.com/an-exploration-of-neural-networks-playing-video-games-3910dcee8e4a
18. https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be
19. https://blogs.unity3d.com/2020/02/28/training-intelligent-adversaries-using-self-play-with-ml-agents/
20. https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a
21. https://medium.com/@nabil_lathif/machine-learning-in-games-development-1c62c9a5ddcf
22. https://medium.com/@markus.x.buchholz/deep-reinforcement-learning-introduction-deep-q-network-dqn-algorithm-fb74bf4d6862
23. https://deepai.org/machine-learning-glossary-and-terms/markov-decision-process
24. https://deepmind.com/research/case-studies/alphago-the-story-so-far
25. https://arxiv.org/pdf/1602.01783.pdf