**Custom Environments**

**What is a Custom Environment**

We have explored training and evaluating various reinforcement learning algorithms to beat scenarios in the unity and vizdoom gym environment adapted from the DOOM game,

A custom environment is a reinforcement learning environment that is created to reflect the unique challenges of a specific scenario(Willies Ogola, Building a Reinforcement Learning Environment using OpenAI Gym, 2022)

An example of a custom environment could be a manufacturing factory floor where a trained agent must map and avoid obstacles, while completing a course to deliver work tools from A to B, this scenario can be simulated sand trained on a computer as a custom environment

Though reinforcement learning and machine learning models in general are known for their highly generalizable nature, training an algorithm, and structuring its reward scheme on an environment replicated to be most like its test destination is likely to produce better results

In this chapter we explore an example scenario where a customer has requested an agent be trained to navigate a custom environment based on a game,

We build a custom reinforcement learning environment using OpenAi Gym.

All code files including learned weights for the model can be found at this link

<https://drive.google.com/file/d/1fFpMW7MBcS_1NmwtRANqqqcMvI91eUtC/view?usp=sharing>

Github: <https://github.com/Jolomi2k9/RL_Game_AI/blob/nicholas/leprechaun_environment.zip>

**Description Of the Environment**

The environment is based on a unity game Capstone2D which was built by our team,

A screenshot of a video game

Description automatically generated with medium confidence

*Capstone2D game capture*

The aim of the game is to navigate a leprechaun to shoot at bugs without colliding with them,

The leprechaun must survive in the environment for as long as possible will shooting as many bugs as possible

1. The leprechaun must survive in the environment for as long as possible
2. The episode terminates if the leprechaun collides with a bug,
3. There are ammo packs which the leprechaun can collect to refill bullets to a fixed number max\_bullets, here we chose 50

**Elements Of the Environment**

There are 5 elements in the environment

* Leprechaun-protagonist of the game and trainable agent
* Bug-antagonist of the game, moves in a straight line down the screen, dies with one bullet
* Ammo- represents a pack of max\_bullets bullets
* Bullet- shown below as the shamrock is the bullet fired by leprechaun
* Bg-the blue stone background image to the game on which every other element is placed

A picture containing text, clipart

Description automatically generated

*The 5 elements*

they are defined by 5 classes, which all inherit from the point base class

**The Point Base Class**

The point base class defines the basic unit of an element in the environment

Text

Description automatically generated

It initializes the element by accepting the allowed coordinates for the element to be within the environment, and defines functions to position and move the element within the environment

Text, letter

Description automatically generated

the individual element classes inherit from the point base class, here the element icon is read

converted to grayscale and resized

**Creation Of the Environment**

The first consideration when designing a custom environment is to determine what the observation and action space will be

* The observation space defines the environment, it can either be discrete or continuous, an example of a discrete observation space would be that of a card game where the environment state is determined by the values of all cards in play, an example of a continuous observation space would be an environment where the agent’s position is determined by real valued coordinates or pixel location on screen

(Jacob Wilson, What is observation space in Openai gym? – Tech-QA.com, 2022)

* The action space can also be either continuous or discrete and defines the possible actions our agent can take, an example of a discrete action space is a simple platformer game where actions correspond to 0:jump, 1:move forward and 2:move backward but the action does not quantify the outcome, an example of continuous action space could be a rotary cannon that shoots golf balls at a target, the shoot action also quantifies how much power the ball is shot with and the left and right actions also quantifies by how many degrees the cannon swivels

(Jacob Wilson, What is observation space in Openai gym? – Tech-QA.com, 2022)

**Lepscape Class**

This class inherits from Gym.env and contains all the code to create and run the environment, it has 3 main functions, \_init\_, reset and step

**\_init\_ function**

We start by defining the observation and action space

The observation shape is a continuous 500 X 500 X 1 canvas where the “1” dimension is a single grayscale colour channel dimension ranging from 0 to 255, using grayscale or black and white imagery as opposed to an RGB image will exponentially reduce training time as the neural network has significantly less information to interpret

All elements in the environment will be drawn on this canvas and the canvas will represent the state of the environment

The Action space is a set of discrete values ranging from 0 to 6 representing respectively the actions

{0: "Right", 1: "Left", 2: "Down", 3: "Up", 4:"shoot", 5: "Do Nothing"}

**Reset Function**

This function is responsible for resetting our environment at the end of each episode, here we redefine all variables, clear the canvas and draw the leprechaun on it for a fresh episode

A picture containing chart

Description automatically generated

*a reset canvas*

The reset function also defines and reset variables that keep track of the environment state

ep\_length : keeps track of the length of the episode in timesteps, one timestep represents one frame of the game in which all elements of the environment take a single action, it is set to “Episode\_length” configuration and here we have chosen steps

bullets\_left: keeps track of how many bullets the leprechaun has left, ranges from 0 to max\_bullets

ep\_return : keeps track of the total reward of the episode

self.bug\_count : number of bugs in this episode

self.ammo\_count :number of ammo packs in this episode

these variables below are returned as information by the step function, they are used to evalueate the models perfomance eventually

ep\_kill\_count: how many bugs were killed in this episode

ep\_bullets\_used: how many bullets were used in this episode

ep\_died: true if the leprechaun died and false if the episode\_length ran out

ep\_ammo\_collected: how many ammo packs was the leprechaun able to collect

**The step Function**

The step function accepts an action applies it to the agent and updates all other elements in the environment, it advances the environment by one frame, at the end of which it returns the environment state, any rewards accrued and whether the episode is done.

We start by asserting that the action is valid and contained within the action space

Once that is done, the action is implemented to the leprechaun as shown below



*Implement action to leprechaun*

New bugs and ammo are spawned at the top of the screen based on their respective probabilities which are 5% and 1% chance respectively

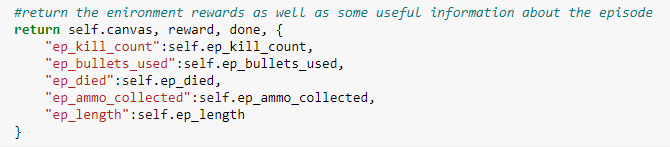
Next, check for collisions, first between bugs and bullets in which case both elements are removed from the environment and rewards are updated

Next, update the rest of the elements, if the leprechaun has collided with a bug the episode is ended, and rewards are updated, else all bugs are moved downward by delta 5

If the leprechaun has collided with an ammo pack the pack is removed, bullets left is reset to max bullets and rewards updated,

All elements that have reached the bottom of the screen are removed

Next, check if the episode has run out of time steps where we end the episode



*Step info*

At the end of the step function, information from the step is returned as shown above

A screenshot of a game

Description automatically generated with medium confidence

*A working environment*

Testing the environment for 10 episodes produced the following result

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Total Kill | Total Bullets Used | Bullets Per Kill | Total Ammo Picked | Episode Reward Mean | Episode Length Mean | Number of Deaths |
| 31 | 435 | 14.0 | 4 | 1.1 | 156.9 | 8 |

**Training**

**Configuration Of the Environment and Reward Structure**

Text

Description automatically generated with low confidence

The environments parameters are stored in a config variable to ensure flexibility in training.

This config variable is used to determine dimensions of the screen, probability of bugs and ammo packs spawning on any given step and the reward for each action.

The step\_reward is the reward for executing a step in the environment. As shown above, has been set to a very low negative number this has the effect of incentivizing the agent to sacrifice living longer for being more aggressive towards shooting at the bugs, there is a large penalty for colliding with the bug and an equally large reward for killing a bug, there is a small penalty for firing a bullet and reward for collecting ammo pack to prevent wastage and consequently running out of bullets and to encourage the agent to collect ammo packs

**Preparation for Training**

A Few actions were taken to prepare the environment for training

The environment is first scaled down from the original size of 800X800X3 to a 500x500X1 canvas to increase the convergence time for the model

all images in the environment are transformed from 3-color channel RGB image to a single channel grayscale image to reduce training time

A virtual machine shown below is setup for training the environment due to the resource intensive nature of reinforcement learning

Graphical user interface, chart, application

Description automatically generated

*Tensorboard running in vm*

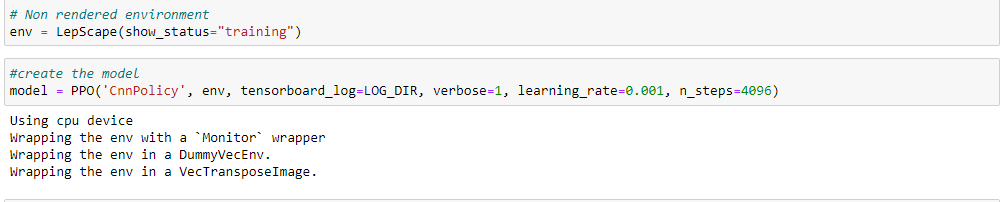
A callback function is set for the model to log training data, and save the weights learned by the model every 10000 timesteps, these training logs will later be used to plot training metrics on tensorboard

A picture containing chart

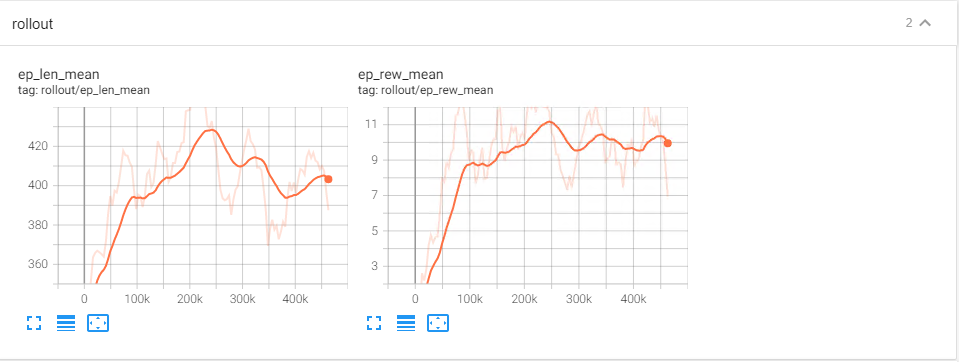
Description automatically generated

*Environment downsized and greyscaled*

The model is then trained with the PPO CNN policy as shown below for 500,000 Timesteps



**Training Process**



*Length and reward mean*

Based on the logs shown above that the agent lived steadily longer in the environment until about timestep 250k where it peaked at 444 out of 500 steps per episode, this tracks well with the increasing rewards, however the model started to sacrifice living longer for being more aggressive by going after the bugs causing it to die early a lot more resulting in a disproportionate decline of the episode length with respect to the episode reward

Explained variance is a measure of how much of the rewards gained in the episode is explained by decisions taken by the model as opposed to random, the value steadily increases to a peak of about 60% at the 450k timesteps

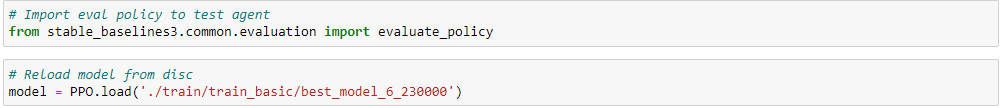
Chart, line chart

Description automatically generated

*Explained variance*

This metric is important because it shows that the model is learning

**Results Metrics**

****

*Model 230k is loaded*

After training completed the environment was tested for 100 episodes, on a random agent and the trained model 230000 agent and the results are shown below

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Total Kill | Total Bullets Used | Bullets Per Kill | Total Ammo Picked | Episode Reward Mean | Episode Length Mean | Number of Deaths |
| Random agent | 296 | 4569 | 15.4 | 30 | 1.5 | 126.8 | 57 |
| Trained Agent | 320 | 3489 | 10.9 | 36 | 8.0 | 100.7 | 43 |

**Reflection on Results**

**Total Kill**: the trained agent killed a lot more bugs than the random agent despite living shorter on average which shows evidence of learning to prioritize kills over just surviving in the environment

**Total Bullets Used**: the trained agent used less bullets than the random agent which shows evidence of conserving bullets and not firing sporadically

**Bullets Per Kill**: this metric is an important one and shows evidence of targeting behaviour, however though the agent uses 5 less bullets than the random agent per kill it is still wasting a lot of bullets, an ideal scenario here would be a 1:1 ratio, training the model for much longer epochs will likely improve the targeting behaviour

**Total Ammo Picked**: the trained agent picked 6 more ammo packs than the random agent despite living for a shorter period on average, the difference is not much but this is not necessarily a bad thing as the model may have learned to only attempt to pick an ammo pack when the shoot action stops producing bullets due to running out of ammo

**Episode Reward Mean**: here we see a clear difference with the random agent which shows that overall, the model is adapting to the aims set out for it with the reward structure.

**Episode Length Mean**: the random agent lived longer on average than the trained agent, further investigation revealed that the reason for this was due to the small negative step reward of -0.001.

The model learned it incurred penalties by simply avoiding bugs, so it would go after them which will result in the leprechaun getting killed due to a combination of sometimes poor targeting, attempting to shoot when it had run out of bullets or shooting accurately at the bug but sometimes the collision does not register due to an unfixed logical error within the underlying check collision code, interestingly in later epochs the model tried to fix this by firing multiple bullets at the bug, the reverse effect of this is that the bullets per kill ratio increases and it incurs multiple penalties for wasting bullets

Overall, a better reward structure might have been to give the agent a small positive step reward for surviving in the environment also the performance can be improved by making all collisions register without exceptions

**Number of Deaths**: on the opposite side of active targeting a lower number of deaths relative to the random agent showed that the leprechaun learned to dodge bugs

**Applications of Custom Environment**

**Automatic Difficulty**

Custom environments can be used to implement automatic difficulty in games by introducing longer trained agents as the players ability improves

The benefits of this are that it automates not just the labour of manually creating an algorithm corresponding to various levels of difficulty in the game, but it also automates the creativity of how these agents increase the challenge to the player

Instead of the developer conceptualizing how to increase the challenge to the player they can rely on patterns learned in the environment by the model over time to gradually improve in complexity and skill

(Watcharasatharpornpong, Nirach & Kotrajaras, Vishnu. (2009). Automatic Level Difficulty Adjustment in Platform Games Using Genetic Algorithm Based Methodology. 10.5176/978-981-08-3190-5\_482.)

**Special Agents**

By manipulating the reward structure in a custom environment, special Agents can be introduced who are incentivized to achieve a particular task in the environment, in the LepScape environment for example a custom “bullet chaser” bug can be introduced and trained to closely follow ammo packs and thus confuse the leprechaun into colliding with the bug to while trying to collect the ammo pack,

While it Is reasonable to assume that a model trained to oppose the leprechaun might learn this behaviour over time, the main aim of designing and training special agents is to explore the creativity of the developer without having to manually implement a new algorithm that codes for the special characters unique behaviour and how it will become more challenging as the players skill Improves

(Sung and Cho, 2012)

References

Engineering Education (EngEd) Program | Section. 2022. *Building a Reinforcement Learning Environment using OpenAI Gym*. [online] Available at: <https://www.section.io/engineering-education/building-a-reinforcement-learning-environment-using-openai-gym/> [Accessed 16 May 2022].

Tech-qa.com. 2022. *What is observation space in Openai gym? – Tech-QA.com*. [online] Available at: <https://tech-qa.com/what-is-observation-space-in-openai-gym/> [Accessed 16 May 2022].

(Watcharasatharpornpong, Nirach & Kotrajaras, Vishnu. (2009). Automatic Level Difficulty Adjustment in Platform Games Using Genetic Algorithm Based Methodology. 10.5176/978-981-08-3190-5\_482.)

Sung, Y. and Cho, K., 2012. A Method for Learning Macro-Actions for Virtual Characters Using Programming by Demonstration and Reinforcement Learning. *Journal of Information Processing Systems*, 8(3), pp.409-420.