Space Cannons



Contents

[Introduction 3](#_Toc108699277)

[Game description 3](#_Toc108699278)

[Preliminaries 3](#_Toc108699279)

[PyGame 3](#_Toc108699280)

[OpenAI Gym 4](#_Toc108699281)

[Custom Environment for OpenAI Gym 5](#_Toc108699282)

[Space Cannon Installation 5](#_Toc108699283)

[Code Description 6](#_Toc108699284)

[SpaceCannons.py 6](#_Toc108699285)

[Params\_game.py 7](#_Toc108699286)

[Calculating cooperative rewards 7](#_Toc108699287)

[Collecting Demonstrations and Human Feedback 7](#_Toc108699288)

[Collecting data 7](#_Toc108699289)

[Resources 8](#_Toc108699290)

# Introduction

Space Cannons is a two-player shooting game designed as a reinforcement learning test bed. Space Cannons is built upon a popular game development library called Pygame and follows the OpenAI gym Framework. The game is developed in a highly customizable manner to accommodate different research needs.

The purpose of this document is to provide a detailed explanation of Space Cannons and its implementation details.

## Game description

In Space Cannons, there are two cannons which can be controlled by one or more agents depending on the implementation. The purpose of the cannons is to shoot at the enemies coming from the top of the screen in order to kill them. Various properties such as enemy health and bullet damage can be adjusted. Moreover, the game supports hybrid AI applications such as inverse and interactive reinforcement learning.

# Preliminaries

## PyGame

Python facilitates developers to increase the confidence and productivity about their developing software from development to deployment and maintenance. The benefits of making Python the perfect solution for machine learning and AI-driven projects include simplicity and consistency, flexibility, access to powerful AI and machine learning (ML) libraries and frameworks, platform independence, and large communities. This is why python was a choice for our research and hence we decided pygame as our game development library. Some of the other reasons why we specifically choose pygame are as follows:

* Python is a programming language with multiple in-built features
* It is easy to utilize and run on any environment
* We can implement pygame by single (long) block of source code
* A lot of individuals who have utilized Pygame before and can support you
* Luckily, you do not have to run Apache on your neighbourhood PC; you can run something lightweight like httpster for most undertakings

## OpenAI Gym

##

OpenAI gym is undeniably the most popular choice for implementing environments to train your agents. OpenAI gym provides many of these environments out-of-the-box. Furthermore, OpenAI gym provides an easy API to implement your own environments.

OpenAI Gym consists of two parts:

* The gym open-source library: This consists of many environments for different problems to test your reinforcement learning algorithms. Some of the categories of environments available are classic control and toy text, Algorithmic, Atari and 2D and 3D robots.
* The OpenAI Gym service: This allows you to compare the performance of your agent to other trained agents.

Open AI Gym has an environment-agent arrangement. It simply means Gym gives you access to an “agent” which can perform specific actions in an “environment”. In return, it gets the observation and reward as a consequence of performing a particular action in the environment.

 

There are four values that are returned by the environment for every “step” taken by the agent.

1. Observation (object): an environment-specific object representing your observation of the environment. For example, the board state in a board game.
2. Reward (float): the amount of reward/score achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward/score.
3. Done (boolean): whether it is time to reset the environment again, such as when you have just lost your last life in the game.
4. Info (dict): diagnostic information useful for debugging. However, official evaluations of your agent are not allowed to use this for learning.

## Custom Environment for OpenAI Gym

OpenAI gym contains over 100 different environments, but if anyone wants to create a new game environment or needs to convert an already available game into an OpenAI gym environment then it is required that the new environment follows the same code flow as OpenAI gym as explained above. This is why the main code of Space Cannons also follows the same code structure consisting of object, reward, done and info. Along with the code structure, OpenAI gym also provides a sample file structure which needs to be followed in your custom environment. Refer to the [official guide](https://github.com/openai/gym/blob/31be35ecd460f670f0c4b653a14c9996b7facc6c/README.rst) for more information.

 

 Figure: file structure

# Space Cannon Installation

The code for Space Cannons can be retrieved from the github repository (<https://github.com/vjmehul/SpaceCannons/tree/main/SpaceCannons>).

The main directory of Space Cannons contains a requirement.txt file which contains information about all the python packages required to execute Space Cannons. You can use “pip install -r requirements.txt” to install all the dependencies.

Once the installation package installation is complete, we need to install the Space Cannon environment in the OpenAI gym library. To do this, go to the folder “SpaceCannons” and execute setup.py using command “python install setup.py” or “pip install -e .”.

After this step you can execute the random\_input.py file in the main game directory to run the game with random steps by both the cannons.

# Code Description

## SpaceCannons.py

SpaceCannons.py is the file containing the class Space Cannons with all the required methods. A small description of these functions is as follows:

* Init: init function initialize the game with the game screen and other variables.
* Reset: This function is executed at the start and end of the game when we need to reset all the counters, pygame sprite groups, player groups and backgrounds.
* Step: The step function executes a single step in the environment and is the most important function. This function takes care of extracting key presses from game and translating them to actions that are taken in the environment. This function also handles collision detection which is then used to calculate rewards. Finally, this function is also responsible for logging experimental data at every step.
* Render: This function is responsible for drawing the python objects at every step of the game.

These following functions use various classes defined in the Game folder to execute the game:

background.py: Creates a background with stars during the game play (NOTE: might contain some memory leak issues).

bullet.py: Extracts bullet images from the Game\_imgs folder and defines two bullet objects (one for each player). It also updates the direction and angle of fire using a simple trigonometric equation.

enemy\_spawner.py: Spawns enemy after every three seconds. This can be changed by updating spawn\_timer variable.

enemy.py: Extracts images of enemies and updates them on the game screen with the code for explosion animation.

healthbar.py and hud.py: Updates the health bar and game status on the screen.

partical.py: Creates particle animation on the game screen when a bullet hits an enemy.

players.py: Initializes player1 and player2 objects with the code to update player positions, shooting actions and health reduction of players.

score.py: Updates scores on the game screen.

## Params\_game.py

In the Params\_game.py file, all the internal game parameters can be changed depending on the use case. This file contains options to change the player health, enemy health, bullet damage, enemy evade penalty, player positions and game difficulty.

# Calculating cooperative rewards

One crucial feature of Space Cannons is that it can provide separate scores to cannon agents when they cooperate to kill an enemy depending on the grade of cooperation displayed by the agents. This is done by counting the number of hits by each enemy and then creating a coop-factor (α) to decide if the hit was cooperative or not (for more info refer to [this](https://www.hhai-conference.org/wp-content/uploads/2022/06/hhai-2022_paper_19.pdf)). If the value of alpha is very high, then even kills with high cooperation will not be considered as cooperative in the game, and vice versa. The coop-factor can be changed in the params\_game.py

# Collecting Demonstrations and Human Feedback

Space Cannons was inherently designed to also collect demonstration data coming from human experts. This data can then again used in the training process of any leaning algorithm.

To collect demonstrations, just enable demo=True in the random\_input.py file. (NOTE: the implementation of this feature is recommended to execute in the main learning algorithm.)

Along with demonstrations, Space Cannons can also incorporate human feedback signals. The code to implement this is written in SpaceCannon.py.

# Collecting data

To collect game statistics, Space Cannons uses the wandb api. Wandb is an online tool to build better models faster with experiment tracking, dataset versioning, and model management. To use wandb follow the following steps.

* 1. Create and login to wandb with your account.
	2. Extract your unique API key from the settings.
	3. Uncomment wandb.init() and wandb.log(info) command in line 53 and 354 in spacecannons.py
	4. Run the game
	5. The game will now ask you for your API key you can paste it in the terminal.
	6. Logging initializes

The above steps are applicable only for the first-time use.

# Resources

<https://www.oreilly.com/library/view/reinforcement-learning-with/9781788835725/5d745161-47f2-41e6-aa9b-a928ba84c346.xhtml>

<https://www.tutorialandexample.com/pygame-pros-and-cons>

<https://medium.com/velotio-perspectives/exploring-openai-gym-a-platform-for-reinforcement-learning-algorithms-380beef446dc>

<https://towardsdatascience.com/beginners-guide-to-custom-environments-in-openai-s-gym-989371673952>