the-mandalorian

Jivitesh Jain 2018101092 Assignment 1 DASS

Introduction

This is a CLI platformer arcade game written in Python 3, heavily inspired by Jetpack Joyride.
Din is a mandalorian living in the post-empire era. He is one of the last remaining members of his clan in the galaxy and is currently on a mission for the Guild. He needs to rescue The Child, who strikingly resembles Master Yoda, a legendary Jedi grandmaster. But there are lots of enemies and obstacles in his way, trying to prevent Din from saving Baby Yoda. Din is wearing classic mandalorian armour and has a jetpack as well as a blaster with unlimited bullets. You've got to help him fight his way through and rescue Baby Yoda.

Gameplay

- You control the Mandalorian throughout his efforts to save Yoda by avoiding or killing fire beams, collecting coins, using powerups and dodging magnets.
- The world has gravity and a horizontal drag force that will make Mandalorian lose his forward inertia and start drifting back with the rest of the objects in the absence of user input. However, Mandalorian never goes out of the viewport.
- The Mandalorian can move forward or backwards and jump up using his Jetpack, in whose absence gravity brings him down.
- He can shoot an unlimited number of bullets, as well as occasionally use a temporary shield.
- Bumping into an obstacle causes the Mandalorian to lose one of the few lives he has, and brings him closer to defeat and Baby Yoda to his doom.
- Magnets attract the Mandalorian's armour, making it difficult for him to control himself, while Boosts speed up the entire game - is this really a powerup?!
- The Mandalorian also has a dragon that he can summon at his will. The dragon stays for a limited time before it gets bored, and is unaffected by any obstacles along the way. Collecting coins is still Mandalorian's job. :P
- Towards the end of the game, Mandalorian must defeat the Boss Dragon. He breathes snowballs directed towards the Mandalorian, and each one of them can cost Mandalorian a life. Bumping into the Boss is a direct stairway to death and his faithful dragon won't be here to help him on this one. What's more, the Boss will kill Baby Yoda if Mandalorian doesn't reach him in time.

Can you help Mandalorian save Baby Yoda?

Controls

- `W`: activate Jetpack
- `A`: move backward
- `D`: move forward
- `E`: shoot
- `G`: summon Dragon
- `SPACE`: activate Shield
- `Q`: quit

Note: The controls are force-based. You do not need to press them continuously to achieve motion. Just like an actual Jetpack :)

The Classes

This game follows the object-oriented programming paradigm. Here's a description of the classes that make up this game:

Game

The `Game` class encapsulates the entire game. It contains methods for *playing* the game (i.e. running the game loop), dynamically building the world and instantiating all other necessary objects, and handling the physics, collisions and the timing.

Screen

The `Screen` class encapsulates the display functionality. It manages a matrix the size of the display, renders objects according to the representation and position they provide and prints the display whenever called upon.

Thing

The `Thing` class represents anything tangible in the game, and is the base class from which several other classes inherit. It encapsulates an object in the game's world, with a position (x and y coordinates), a representation, a velocity and an accelaration. It has several methods, such as those for moving objects according to their position, velocity and accelaration, showing objects, checking if objects are out of the screen etc.
Every `Thing` object, by default has a backward velocity (to give the illusion of the viewport moving forward), and remains unaffected by gravity. These characteristics can be overridden whenever required.

FireBeam

The `FireBeam` class inherits the `Thing` class and represents the fire beam obstacles.

Coin

The `Coin` class inherits the `Thing` class and represents the coins that appear on screen.

MandalorianBullet

The `MandalorianBullet` class inherits the `Thing` class and represents the bullets that Mandalorian fires. Several aspects of the `Thing` class have been overridden to model the substantially different physics of the bullets.

BossBullet

The `BossBullet` class inherits the `Thing` class and represents the snowballs fired by the Boss enemy. It contains methods that direct its initial velocity towards the Mandalorian.

Boost

The `Boost` class inherits the `Thing` class and represents the Boost powerup that appears during the game. It contains methods that apply the effect to each of the objects in the game.

Magnet

The `Magnet` class inherits the `Thing` class and represents the Magnets that appear during the game. It has functions for calculating the forces on objects to attract them towards the magnet.

Mandalorian

The `Mandalorian` class inherits the `Thing` class and represents the Mandalorian himself. It contains methods for moving the Mandalorian according to user input, keeping the Mandalorian on the screen at all times, implementing a *drag-like* force in the absence of a forward user input among other things. The default functionality of the `Thing` class has been overridden to implement gravity and remove the continuous backwards velocity.

Dragon

The `Dragon` class inherits the `Thing` class and represents the dragon that can be summoned by the Mandalorian. It has methods and attributes that help generate the wriggle effect. It overrides several of the functionality of the `Thing` class to model the different physics of the dragon.

Object Oriented Concepts

Here's a list of the instances (bad pun :P) of the OOPS concepts in use:

Inheritance

The `Mandalorian` , `Boss` , `Dragon` , `MandalorianBullet` , `BossBullet` , `Coin` , `FireBeam` , `Boost` , `Magnet` classes all inherit the `Thing` class.

Polymorphism

The `move()` method defined in the `Thing` class has been overriden in the `Mandalorian` class. This example of *method overriding* represents polymorphism. Several other examples are dotted throughout the code base.

Encapsulation

The entire game is modelled using classes and objects which encapsulate logically different entities.

Abstraction

Methods like `Thing.move()` , `Thing.show()` , `Screen.add()` , `Game.play()` , `Game.handle_collisions()` abstract away the details and make the code more readable and organised.

Get Set Go

To install the game:

1. Clone this repository

   ```
   $ git clone https://github.com/jiviteshjain/the-mandalorian.git
   ```

   or get the source code any other way.

2. Navigate into the repository

   ```
   $ cd path/to/repository
   ```

3. Set up and activate virtual environment (optional)

   ```
   $ python3 -m venv env
   $ source env/bin/activate
   ```

4. Install the required packages

   ```
   $ pip3 install -r requirements.txt
   ```

5. May the Force be with you!

   ```
   $ python3 play.py
   ```