

Assignment 2

Introduction

About academic integrity

Task overview & download

Task 1: Star War

Task 2: Demonstrating Advantages of Dynamic Typing

Task 3: Duck Typing

Task 4: Demonstrating Advantages of Duck Typings

Sample output & grading

Written report

Submission & deadline

Prepared by Zhisheng Hu and
Yan Liu

CSCI3180 Assignment 2: Star War



Star War

Introduction

In this assignment, you are going to write a python program to implement the rules of a single-player game named *Star War* based on the given C++ implementation. Moreover, we will extend the game and you need to implement more classes for the game. *Note: All the Python implementations in this assignment should be built using any Python version after (including) 3.6.*

The main objectives of this assignment are three-fold for you:

- To practice Python and learn the differences between Python and C++.
- To explore the pros and cons of dynamic typing in Python.
- To better understand duck typing in Python and know how to properly use it.

About academic integrity

We value academic integrity very highly.

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters.
- The penalty (for BOTH the copier and the copiee) may not be just getting a zero in your assignment. There may be additional disciplinary actions from the university, upto and including expulsion.
- Do NOT copy or look at the assignment code/work of another student/person.
- Do NOT share your assignment code/work to anyone and do NOT post it anywhere online.
- While we encourage discussion among students, you should write the code and finish the work on your own. In the discussion, you should NOT go into the details such that everyone will end up with the same code/work.
- You are responsible to double-check your own submission and do your best to avoid "accidents".

Read <https://www.cuhk.edu.hk/policy/academichonesty/> for details of the CUHK policy on this matter.

Task overview & download

You have two main tasks in this assignment.

1. **Coding:** First, download the starting code [HERE](#), which contains the C++ implementation `starWar.cpp` and the python skeleton code `starWar.py` for **Task 1**. Your programming task is to read the task description, understand the given C++ code, and complete the incomplete functions marked with TODO in the given python skeleton code files. Second, download the extension code [HERE](#), which contains the C++ implementation `starWar_enhanced.cpp` and the python skeleton code `starWar_enhanced.py` for **Task 3**. Also, your programming task is to read the task description, understand the given C++ code, and complete the incomplete functions marked with TODO in the given python skeleton code files. You only need to modify and submit two files: **starWar.py** and **starWar_enhanced.py**. Basically, all other files should not be submitted or modified. We will provide the specifics in the [Submission & deadline](#) section.
2. **Writing:** Write a report on dynamic typing and duck typing pros/cons, and save it as a PDF file. We will provide the specifics in the [Written report](#) section.

Read the pinned assignment 2 FAQ post on [Piazza](#) for some common clarifications. You should check that a day before the deadline to make sure you don't miss any clarifications, even if you have already submitted your work then.

If you need further clarifications of the requirements, please feel free to post on the Piazza with the [assignment2](#) tag. However, to avoid cluttering the forum with repeated/trivial questions, please do read all the given code, webpage description, sample output, and latest FAQ carefully before posting your questions. Also, please be reminded that we won't debug for any student's assignment for the sake of fairness and that you should not share/post your solution code anywhere.

Task 1: Star War

This is a small programming exercise for you to get familiar with Python, which is a dynamically typed language. In this task, you have to strictly follow our proposed OO design and the game rules for "Star War" stated in this section.

Description

Star War is a single-player game. When the game starts, the player uses the 4 keys ('w', 'a', 's', and 'd') to move a plane around the space, represented by a 16 x 48 map. **After inputting the keys, you should press the "Enter" key to proceed.** If the key 'w' is pressed, the plane will move up in the next step. If the key 's' is pressed, the plane will move down in the next step. If the key 'a' is pressed, the plane will move left in the next step. If the key 'd' is pressed, the plane will move right in the next step. If you press any key other than these four keys, the plane will keep moving in the previous direction.

Player controls a small plane represented by a 2 x 5 grid, capable of withstanding a maximum of 2 attacks. The enemy controls a large plane represented by a 3 x 7 grid, capable of withstanding a maximum of 5 attacks. Both player and enemy-controlled plane automatically fire a bullet at regular intervals. The enemy moves randomly left and right in the upper area of the map, while the player needs to maneuver their plane to dodge incoming bullets. The objective is to destroy the enemy plane before the player's own plane is destroyed.

Preparation

Input Specification

In this exercise, you are required to use the command line to get input information. There is only one thing requiring user-input, which is the moving direction of the plane. Input one of the 4 pre-defined keys ('w', 'a', 's', and 'd') and press "Enter" to control the direction.

```

      $$$$$$
      $$$$$$
      $$$$$$

      ^^^^^
      ^^^^^

Player: No one can defeat me!

-----
ENEMY   HP: 5
PLAYER  HP: 2
Please select the moving direction! (w : up, s : down, a: left, d: right)
█

```

Move and Display Game Information

The player's plane always starts from the bottom-center of the map. You can use 4 keys ('w', 'a', 's', and 'd') on the keyboard to control the movement of the plane ('w' means up, 's' means down, 'a' means left, 'd' means right. Left is the initial value.). If an invalid key (a key not in the 4 pre-defined keys, e.g., 'q') is pressed, the plane will move in the previous direction.

The planes will automatically fire a bullet at regular intervals. The player's bullets will move upwards, while the enemy's bullets will move downwards.

We call each picture printed by the game a **frame**, and each frame mainly consists of 3 parts, **game map**, **dialogue**, **game state**.

- The planes controlled by the player and enemy, as well as the bullets fired by the planes, will be drawn on the game map area.
- Each frame in the game will randomly select the player or enemy to deliver a line and display it in the dialogue area.
- In the current game, the game state only contains the characters' life value (HP), which will be printed in the game state area.

The player's plane is represented by a 2 x 5 matrix of symbol '^', and the enemy's plane is represented by a 3 x 7 matrix of symbol '\$'.

With each frame transition, all planes and bullets can only move a distance of one grid cell (one grid cell = one symbol or whitespace on the map).


```
-----
          $$$$$$
          $$$$$$
          $$$$$$ ^
            $
              ^
            $
          ^
        ^^^^^
        ^^^^^

Player: No one can defeat me!
-----
ENEMY   HP: 5
PLAYER  HP: 0
Sad! You made sacrifices in this fierce battle.
```

```
-----
          $$$$$$
          $$$$$$
          $$$$$$

              $

          ^^^^^
          ^^^^^

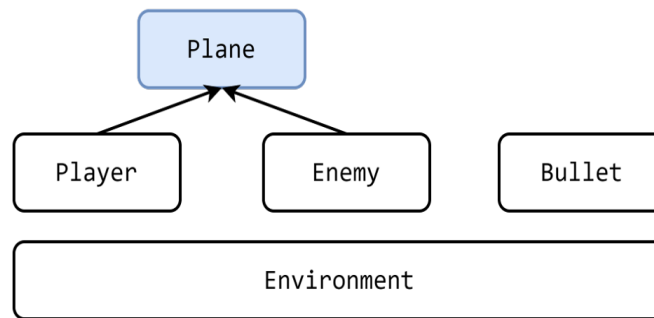
Player: No one can defeat me!
-----
ENEMY   HP: 0
PLAYER  HP: 2
Congratulations! You have defeated them.
```

Class Design for Basic Functions in Python

Please follow the classes defined below in your implementation. You are not allowed to add any class variables, class methods, or classes. We will start the program with `python starWar.py`. The following class design uses the naming convention of Python. In particular, underscored variables are private. You are also required to use property decorator for several attributes in your Python implementation. Please accomplish all tasks marked with TODO in the provided Python template.

The Plane class

It is the base class for the player's and the enemy's plane. We only have 3 classes in this class hierarchy and **Plane** sits at the top. To give you an overview of the class design, here is the class diagram:



Details will be provided below. For anything that appears to be unclear to you, you should refer to the given code. First, let's look at the base class: the **Plane** class.

Data members (Attributes)

```
self._life
```

This is the remaining life value (i.e., HP) of the current plane.

```
self._location, self._size
```

Location (row, col) represents the position of the top-left corner of the plane's rectangle, 'row' denotes the vertical coordinate, 'col' represents the horizontal coordinate. On the other hand, size (height, width) specifies the height and width of the plane's rectangle, 'height' indicates the vertical length, and 'width' refers to the horizontal width.

```
self._symbol
```

This is the symbol of the plane, i.e., '^' represents player, '\$' represents enemy.

```
self._direction
```

This represents the current direction of the plane, defined by the **enum DirectionType**. For example, 0 represents upwards, and 1 represents downwards.

Member functions (Methods)

```
__init__
```

This is the constructor which is responsible for the initialization of the data

members.

```
draw
```

This is the function to print the plane's rectangle to the game map.

```
move
```

This is the function for the plane to update its state for the next frame of the game. For example, move to the next location according to current direction.

```
is_collision
```

This is the function for the plane to check whether the plane has been hit by a bullet.

```
shoot
```

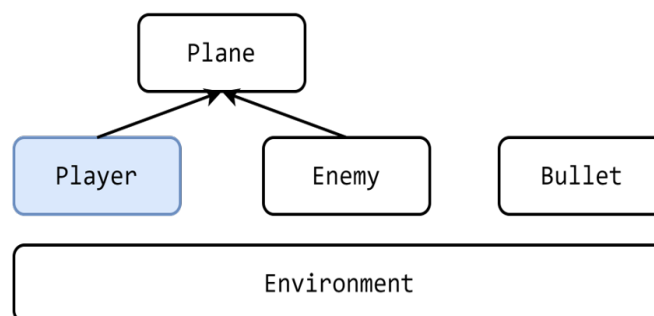
This is the function for the plane to fire a bullet.

```
hit
```

This is the function responsible for the logic when the plane is hit by a bullet, such as decreasing the plane's HP.

The Player class

It is the class of player's plane, a subclass of `Plane`.



Data members (Attributes)

```
self._shoot_interval
```

This indicates the time elapsed since the plane last fired a bullet. When it exceeds a certain threshold, a new bullet will be fired, and this value will be reset to zero.

Member functions (Methods)

```
__init__
```

This is the constructor responsible for the initialization of the data members in Player class.

```
move
```

Override the function of the base class (Plane) to update plane's state for the next frame of the game.

- The player's plane moves to the next location according to current direction.
- The player automatically fires a bullet after a short interval.

```
shoot
```

Override the function of the base class (Plane) to fire a bullet.

```
speak
```

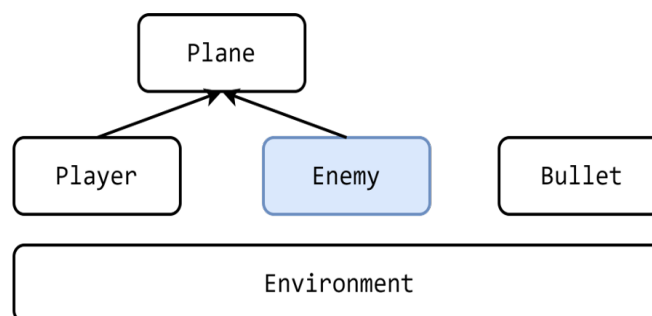
This is the function to print a dialogue line for the player.

```
display_info
```

This is the function to print some information after the player wins.

The Enemy class

It is the class of enemy's plane, a subclass of **Plane**.



Data members (Attributes)

```
self._shoot_interval
```

This indicates the time elapsed since the plane last fired a bullet. When it exceeds a certain threshold, a new bullet will be fired, and this value will be reset to zero.

Member functions (Methods)

```
__init__
```

This is the constructor responsible for the initialization of the data members in Enemy class.

```
move
```

Override the function of the base class (Plane) to update enemy's state for the next frame of the game.

- The enemy's plane moves to the next location according to current direction.
- The enemy automatically fires a bullet after a short interval.
- The enemy has a 10% chance of changing its direction in each frame.

```
shoot
```

Override the function of the base class (Plane) to create and fire a bullet forward.

```
speak
```

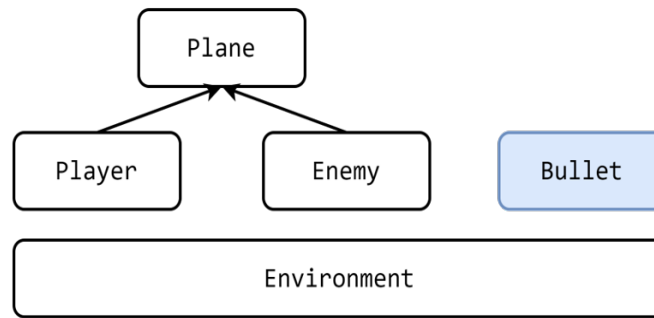
This is the function to print a dialogue line for the enemy.

```
display_info
```

This is the function to print some information after the enemy wins.

The Bullet class

It is the class for the bullets.



Data members (Attributes)

```
self._validity
```

This boolean variable determines the validity of the bullet. If the bullet reaches the map boundaries or collides with an opponent's plane, it becomes invalid and will be removed in the next game frame.

```
self._bullet_type
```

This variable represents which plane fired this bullet. For example, if this bullet is from the player, then it is `BUL_FROM_PLAYER` type.

```
self._bullet_on_edge
```

This boolean variable represents whether this bullet has reached the map boundaries.

```
self._location, self._symbol, self._direction
```

These three variables are basically the same as in the `Plane` class.

- The `location` represents the position of the bullet.
- The `symbol` represents the symbol of the bullet, e.g., '^' represents player's bullet, '\$' represents enemy's bullet.
- The `direction` represents the current direction of the bullet, defined by the `enum DirectionType`. For example, 0 represents upwards, and 1 represents downwards.

Member functions (Methods)

```
__init__
```

This is the constructor responsible for the initialization of the data members in Bullet class.

```
draw
```

This is the function to print the bullet to the game map.

```
check_on_edge
```

This is the function to check whether this bullet has reached the map boundaries.

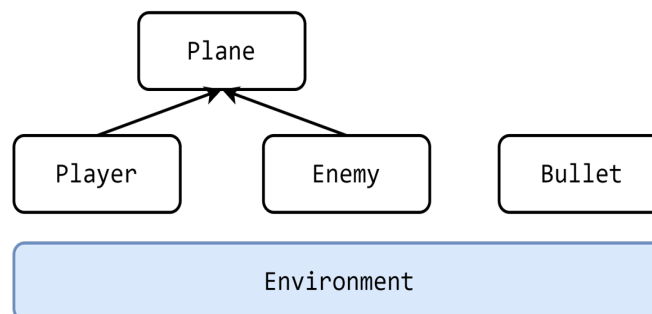
```
move
```

This is the function for the bullet to update its state for the next frame of the game.

- The bullet moves to the next location according to current direction.
- Check for collision with the opponent's plane.
- Check if the bullet has reached the boundaries of the map.

The Environment class

This class is about the game environment.



Data members (Attributes)

```
self._speaker, self._winner
```

The **speaker** represents the character currently displaying the dialogue. The **winner** represents the character who has won the game.

Member functions (Methods)

```
__init__
```

This is the constructor responsible for the initialization of the game environment.

`move_all`

This function calls the `move()` function of all planes and bullets, representing the transition of the entire game state to the next frame.

`draw_all`

This function calls the `draw()` function of all planes and bullets, rendering all these game objects onto a game map.

`check_state`

This function checks the current state of the game and performs corresponding operations, such as removing bullets with a validity of false and checking if the game meets the conditions for ending.

`display_all`

This function prints the game map, dialogue, and game state information sequentially on the screen, effectively displaying one frame of the game.

Please note that the previous `draw_all()` function only generates the data for the game screen but does not print it.

`display_result`

This function is used to print some information when the game ends.

`get_input`

This function is used to receive player's keyboard input.

`run`

This function is the fundamental running function of the game. It executes in a loop until the game ends and, in each iteration, performs the following tasks sequentially:

- `get_input()`: Waits for and accepts player keyboard input.
- `move_all()`: Advances the game to the next frame state.
- `draw_all()`: Draws the game screen for the next frame.
- `display_all()`: Displays the game screen for the next frame.
- `check_state()`: Checks the game state to determine if the game should

end.

Task 2: Demonstrating Advantages of Dynamic Typing

^xWorry about memory allocation or leakage handled in runtime by system

Dynamic typing is an important characteristic of Python. It has many commonly-claimed advantages:

1. Don't need to declare variables or their data type before assignment.
2. It makes functions more reusable because functions can be applied on arguments of different types.
3. ...

However, type checking can only be carried out at runtime, incurring time overhead and reliability issues.

Please provide at least one concise example code with explanation in the context of this assignment to demonstrate the advantages of Dynamic Typing and at least one concise example code with explanation in the context of this assignment to demonstrate the disadvantages of Dynamic Typing.

Task 3: Duck Typing

In this task, you are required to implement another new class `Gift`. After this exercise, you may have a deeper understanding of a special feature of Python called **Duck Typing** that is available only in dynamically-typed programming languages. Also, the C++ code for this new class is provided (`starWar_enhanced.cpp`) and you have to reimplement the new class in Python (a skeleton provided in `starWar_enhanced.py`) based on the following description and the given functions.

Duck Typing

The following synopsis of Duck Typing is summarized from:

- https://en.wikipedia.org/wiki/Duck_typing
- <http://www.sitepoint.com/making-ruby-quack-why-we-love-duck-typing>

In standard statically-typed object-oriented languages, objects' classes (which determine an object's characteristics and behavior) are essentially interpreted as the objects' types. Duck Typing is a new typing paradigm in dynamically-typed (late binding) languages that allow us to dissociate typing from objects' characteristics. It can be summarized by the following motto by the late poet James Whitcombe Riley:

When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

The basic premise of Duck Typing is simple. If an entity looks, walks, and quacks like a duck, for all intents and purposes it is fair to assume that one is dealing with a member of the species *anas platyrhynchos*. In practical Python terms, this means that it is possible to try calling any method on any object, regardless of its class. An important advantage of Duck Typing is that we can enjoy polymorphism without inheritance. An immediate consequence is that we can write more generic codes which are cleaner and more precise.

Description

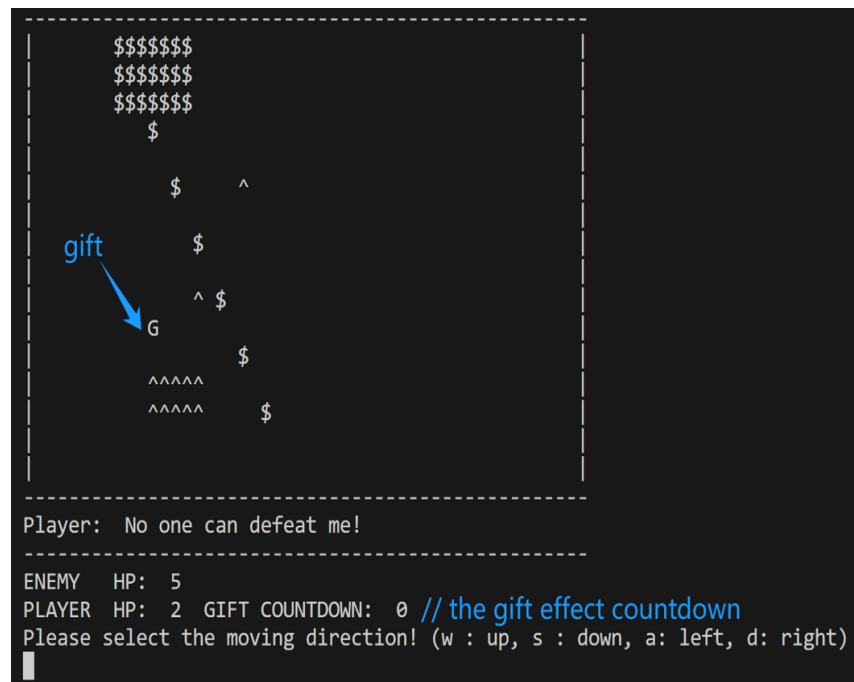
The game is getting harder! The enemy attacks have become more intense, with the enemy planes now firing bullets at a rate of one every 2 frames, instead of one every 5 frames. In this relentless bullet storm, the player has little chance of winning. To address this, **the Jedi Warrior** has provided us with a gift called **the Force**. Whenever the player picks up a gift, the plane enters a **Force Awakening** state that lasts for 30 frames. During this state, the player's plane becomes immune to enemy bullets. Therefore, in order to win, the player must pick up a gift before the enemy bullet storm arrives and collect the next gift before the Force Awakening state ends. Only then can we defeat the enemy and save the universe!

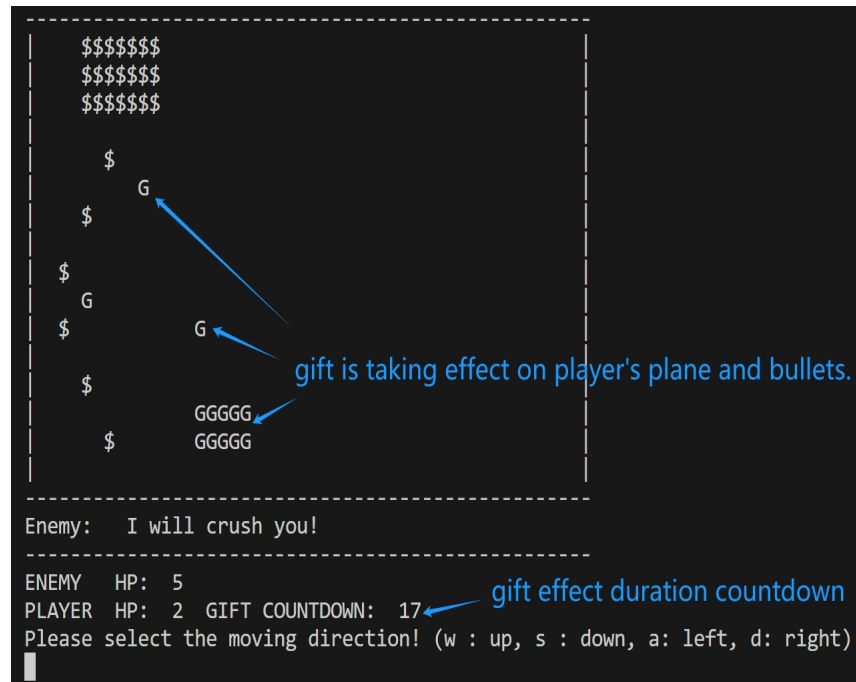
Displaying Game Information

The positions of the gift will be shown on the map, represented by a symbol 'G'. Additionally, the game state area will display a countdown for the remaining duration of the gift effect. You are required to implement the new class and ensure the correct information output.

After picking up a gift, both the player's plane and its newly created bullets will change from the symbol '^' to 'G'. When the gift expires, it will change back from the symbol 'G' to '^'.

The initial position of the gift is randomly generated and can be located at any position on the map. Additionally, after being picked up by the player, the gift will respawn at a new random position in the next frame.



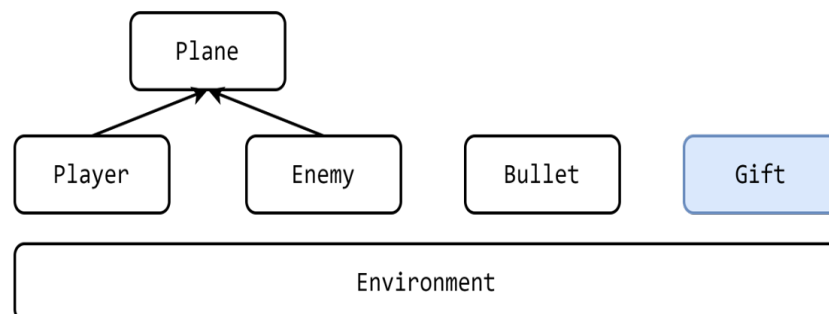


Class Design for Extended Functions in Python

In the extended version, we need to add another new class——**Gift**. Please accomplish all tasks marked with **TODO** in the provided Python template.

The Gift class

It is the class for the gift.



Data members (Attributes)

```
self._validity
```

This boolean variable determines the validity of the gift. If the gift is picked up by the player's plane, it becomes invalid and will be respawn in the next frame.

```
self._location, self._symbol
```

These two variables are basically the same as in the `Bullet` class.

- The `location` represents the position of the gift.
- The `symbol` represents the symbol of the gift, i.e., 'G'.

Member functions (Methods)

```
__init__
```

This is the constructor responsible for the initialization of the data members in Gift class.

```
draw
```

This is the function to print the gift to the game map.

```
move
```

This is the function for the gift to update its state for the next frame of the game. For example, check for collision with the player's plane. If a collision occurs, trigger the gift effect and set the gift's validity to false.

```
respawn
```

This is the function for the gift to respawn at a random location on the map.

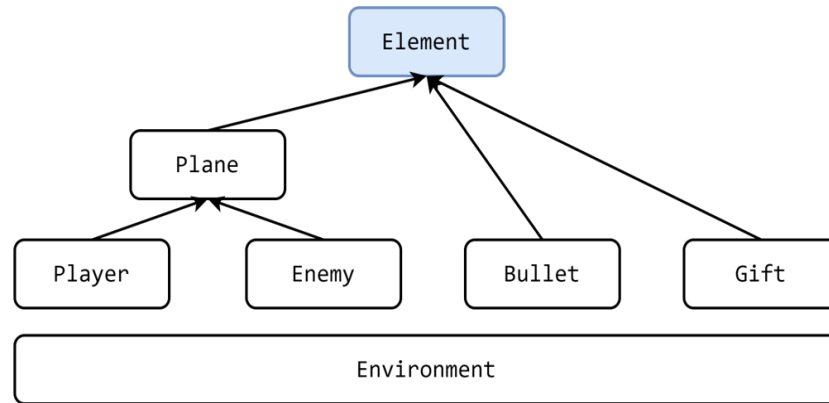
Other classes also have some small changes applied so that they are compatible with to the new class. The main differences are the introduction of the `self.gift_countdown` variable and the `give_gift` function in Player class and the change of `__init__`, `move_all`, `draw_all`, and `check_state` function in the Environment class. Most other parts of the other classes remain unchanged. Please carefully check the changes between `starWar.py` and `starWar_enhanced.py`. Please also carefully check the differences in our C++ implementations between `starWar.cpp` and `starWar_enhanced.cpp` for reference.

Class Design for Extended Functions in C++

In the C++ code, we add another class called Element in addition to the Gift class. It is the abstract base class of all the game object classes (i.e., Plane, Bullet, Gift classes), as shown in the following figure. Due to different properties of different programming languages, we don't have this class in Python.

The Element class

It is the base class for each game object on the game map.



Member functions

`draw`

This function prints the game object to the game map.

`move`

This function updates the game object's state for the next frame of the game.

`get_validity`

This function returns the validity of the game object, with a default value of true.

Task 4: Demonstrating Advantages of Duck Typing

There are at least two places in Task 3 where we use duck typing in Python. Can you identify them? Please provide the scenarios and concise codes to compare your Python implementation and our C++ implementation and explain how duck typing makes coding more flexible and convenient.

Sample output & grading

Your finished program should produce the same output as the following screenshots for each step. But you don't need to worry about the printing format, we have provided the printing functions. What you need to do is to implement some important functions and make the generated information correct in each step.

For Task 1:

First, in each step, your plane should be able to move using the pre-defined 4 keys

('w', 'a', 's', and 'd'). Related information are also correctly printed.

```
-----
                    $$$$$$
                    $$$$$$
                    $$$$$$

                    ^^^^^
                    ^^^^^

-----
Player: No one can defeat me!
-----
ENEMY   HP: 5
PLAYER  HP: 2
Please select the moving direction! (w : up, s : down, a: left, d: right)
█
```

Second, After the player's or enemy's plane is hit by a bullet, their corresponding HP should be correctly reduced by one.

```
-----
    $$$$$$
    $$$$$$
    $$$$$$
      $      ^

          $ ^

          ^ $
          ^^^^^
          ^^^^^

-----
Player: I'm ready for action!
-----
ENEMY   HP: 5
PLAYER  HP: 2
Please select the moving direction! (w : up, s : down, a: left, d: right)
█
```

```
-----
  $$$$$$
  $$$$$$
  $$$$$$      ^
    $
      ^
      $
      ^
      ^^^^^
      ^^^^^
-----
Player:  No one can defeat me!
-----
ENEMY   HP:  5
PLAYER  HP:  1
Please select the moving direction! (w : up, s : down, a: left, d: right)
█
```

Third, you lose the game and the game is over when the HP of player's plane is reduced to 0. Your game should stop and the related information should be correctly printed.

```
-----
  $$$$$$
  $$$$$$
  $$$$$$
    $
      ^
      $
      ^
      ^^^^^
      ^^^^^
-----
Enemy:   I will crush you!
-----
ENEMY   HP:  4
PLAYER  HP:  0
Sad! You made sacrifices in this fierce battle.
```

Fourth, you win the game and the game is over when the HP of enemy's plane is reduced to 0. your game should stop and the related information should be correctly printed.

```

      $$$$$$
      $$$$$$
      $$$$$$

      ^      $

      ^^^^^
      ^^^^^

Player:  No one can defeat me!

-----
ENEMY   HP:  0
PLAYER  HP:  1
Congratulations! You have defeated them.

```

For Task 3:

First, your code should be able to generate the gift correctly according to rules we have mentioned above.

```

      $$$$$$
      $$$$$$
      $$$$$$

      G

      ^^^^^
      ^^^^^

Player:  No one can defeat me!

-----
ENEMY   HP:  5
PLAYER  HP:  2  GIFT COUNTDOWN:  0
Please select the moving direction! (w : up, s : down, a: left, d: right)

```

Second, when the player picks up a gift, its plane and newly created bullets should change to symbol 'G', and the gift countdown starts from 30. And in the next frame, a new gift should respawn at a new random position.

```
-----
$$$$$$
$$$$$$
$$$$$$
$
      ^
    $
      $ ^
      $
    GGGGG
    GGGGG $
      $
      $
-----
Player: I'll save the world!
-----
ENEMY  HP: 5
PLAYER HP: 2 GIFT COUNTDOWN: 30
Please select the moving direction! (w : up, s : down, a: left, d: right)
█
```

```
-----
$$$$$$
$$$$$$
$$$$$$
$
      ^
    $
      $ ^
      $
    G
  G
  $
GGGGG
GGGGG $
      $
      $
-----
Enemy: I will crush you!
-----
ENEMY  HP: 5
PLAYER HP: 2 GIFT COUNTDOWN: 28
Please select the moving direction! (w : up, s : down, a: left, d: right)
█
```

Third, when the gift countdown reaches 0, the player's plane and newly created bullets should change back to the symbol '^'.

you can focus on writing the Python codes without concern for this inconsistency.


Written report

Write a report to comment on the pros and cons of dynamic typing and duck typing in one PDF file named exactly **a2.pdf**, with at most four A4 pages, containing two main sections:

1. Provide example code and necessary elaborations in the context of this assignment for demonstrating advantages and disadvantages of Dynamic Typing as specified in Task 2.
 2. Provide example code and necessary elaborations in the context of this assignment for demonstrating advantages of Duck Typing as specified in Task 4.
-

Submission & deadline

Submission deadline: 23:59:00, Mar 12 (Tuesday).

- Please submit two python files and one PDF file only: **starWar.py**, **starWar_enhanced.py**, and **a2.pdf** ONLY.
- Make sure you have completed the *declaration* at the top of the **starWar.py** file.
- Make sure your two submitted files can be compiled and run with no errors under our official grading environment (see the [Sample output & grading](#) section).
- Do NOT zip the files. Submit them individually in the same submission. Follow these steps:
 1. Go to the CSCI3180 page on [Blackboard](#).
 2. Go to "Assignment Submission" on the left-hand side menu.
 3. Pick "Assignment 2."
 4. For each of the files that you need to submit, click "Browse Local Files", and then browse for it.
 5. The files will be added to the "Attached files" list. Please **check carefully their filenames**. They must be exactly the same as the ones specified. Wrongly named files won't be graded as most grading are automatic.
 6. Click the blue "Submit" button at the bottom of the page.
 7. Now you **must verify** all of your submitted files by downloading them back and make sure they are your own and the latest version. You can download each of the files with the download button . Click "Start New" to submit again if you need to modify your submission.
- You can submit as many times as you want, but **only the latest submission attempt will be graded**. You do NOT get to choose which version we grade.
- If you submit after the deadline, late penalty will be applied according to the submission time of your last submission attempt.
- **Submit early to avoid any last-minute problem**. Only Blackboard submissions will be accepted. Again, you can submit as many times as you want, so you may make an early submission first well before the deadline, and update it later.

- Late submission penalty is 1 point per 5-minutes late. Less than 5-minutes late is considered as a full 5-minutes late. However, your score will be lower-bound by 0 after any mark deduction.
-