# COMP222 Game Principle

Plane VS. Asteroids (Unity Engine)

AI function Implementation

Report

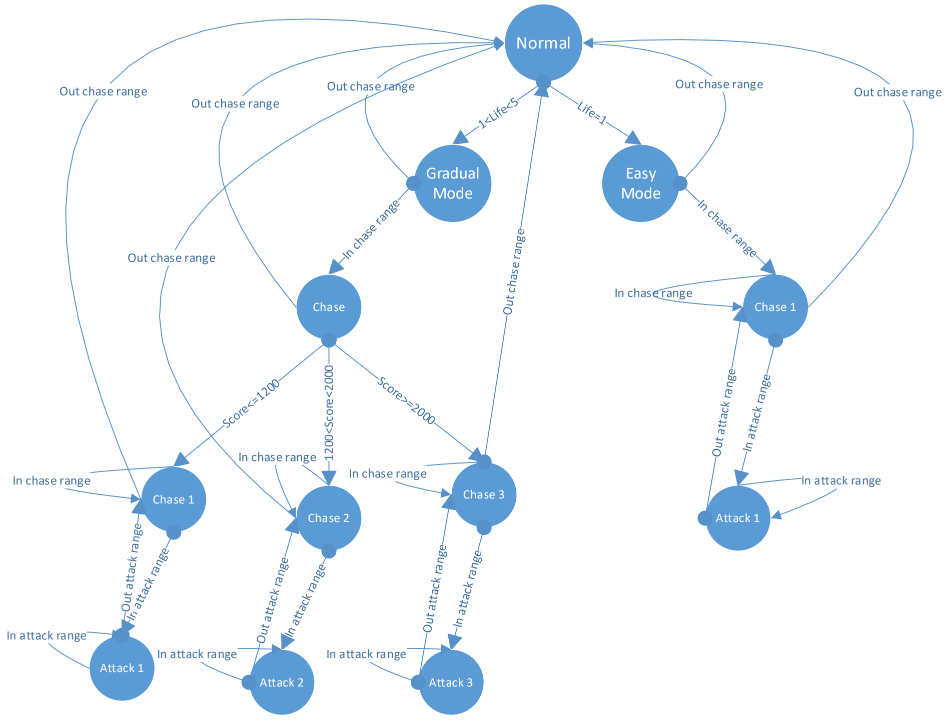
## 1 Introduction

Based on the first version of ‘Plane VS. Asteroids’, this second version of game adds the AI function into the asteroids to make the game more playable and adjust the difficulty automatically. This report will give the state machine description and the detail of the code. The operation of this game is as same as the old version: “A” and “D” for rotations in left and right; “W” and “S” for move forwards and backwards; “C” and “V” for descend and ascend; “Space” or “Left click” for shooting. And all the old features like: floating, collision detection, collision reflection, asteroid split, shooting, collide particle effects, reborn are all kept in this version. And apart from AI, added one new function in the game, which will also mentioned in this report.

## 2 State machine illustration

I choose the finite state to show the logic of AI. There are two reasons for it.

Firstly, this game logic is not that complex and the state is really finite. So, no need to use other models like behaviour tree. Secondly, finite state machine can show the transformations between these states more clearly than other models. I choose to focus on the different transitions among these limited states rather than complex states themselves.



Chase 1
Attack 1(D1): (Easy Mode)

Chase range: 4
Attack range: 1
Bullet Speed: 2000
Chase speed: 0.5
Add Attack speed: 1.5

Chase 2
Attack 2(D2):

Chase range: 7
Attack range: 3
Bullet Speed: 500
Chase speed: 2
Add Attack speed: 3

Chase 3
Attack 3(D3):

Chase range: 10
Attack range: 4.5
Bullet Speed: 300
Chase speed: 4
Add Attack speed: 5

Figure 1. State machine model

In this game, there are two overall modes**: 'gradual mode' and 'easy mode'** and three difficulty combinations: **'Chase 1 Attack 1'(D1), 'Chase 2 Attack 2'(D2) and 'Chase 3 Attack 3'(D3).**

Gradual mode can provide the gradual difficulty transitions when player has more than 2 lives. And when player only have one life left, which means that the game may be too hard for player to finish, then the easy mode will be activated, change and keep the difficulty to easiest level until the game ends.

When in gradual mode, D1, D2 and D3 will be activated according to the score. When $Score \le 1200$, D1 come into function; when $1200 < Score < 2000$, D2 will be activated and when $Score \ge 2000$, highest difficulty D3 will work.

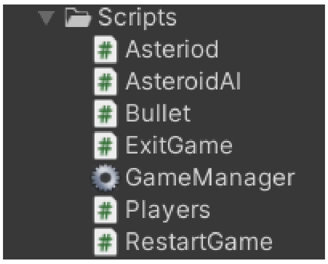
In D1, D2 and D3, there are three actions for the asteroids: normal, chase and attack.

Chase means that when player is in the chase range, the asteroids will change the direction to the player and move towards player at the Chase Speed. When player enter into the attack range, asteroids will suddenly increase the speed and try to crash into the player. This is what I called Attack. The normal refers to all asteroid behaviour in the old version including floating, reflection and split etc.

What is more, each difficulty combination has different chase range, attack range, chase speed and attack speed. And all the parameters are written in the right side of the figure 1.

## 3 Codes demonstrations

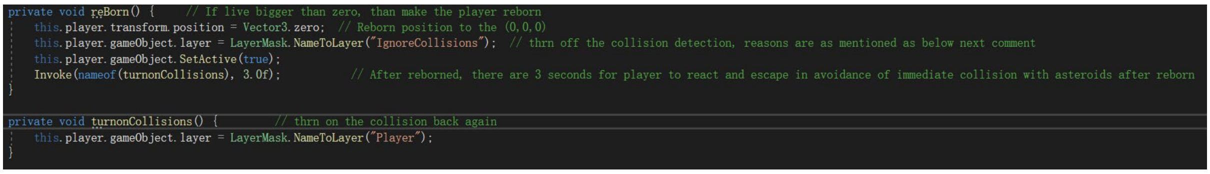
The logic and control of the whole game is consisted seven scripts. Four of them is as same as the old version: Asteroid, Bullet, ExitGame, Players, RestartGame.  And I updated the GameManager and write a new script called AsteroidAI.



Scripts
- Asteriod
- AsteroidAI
- Bullet
- ExitGame
- GameManager
- Players
- RestartGame

Figure 2. Seven scripts in the project

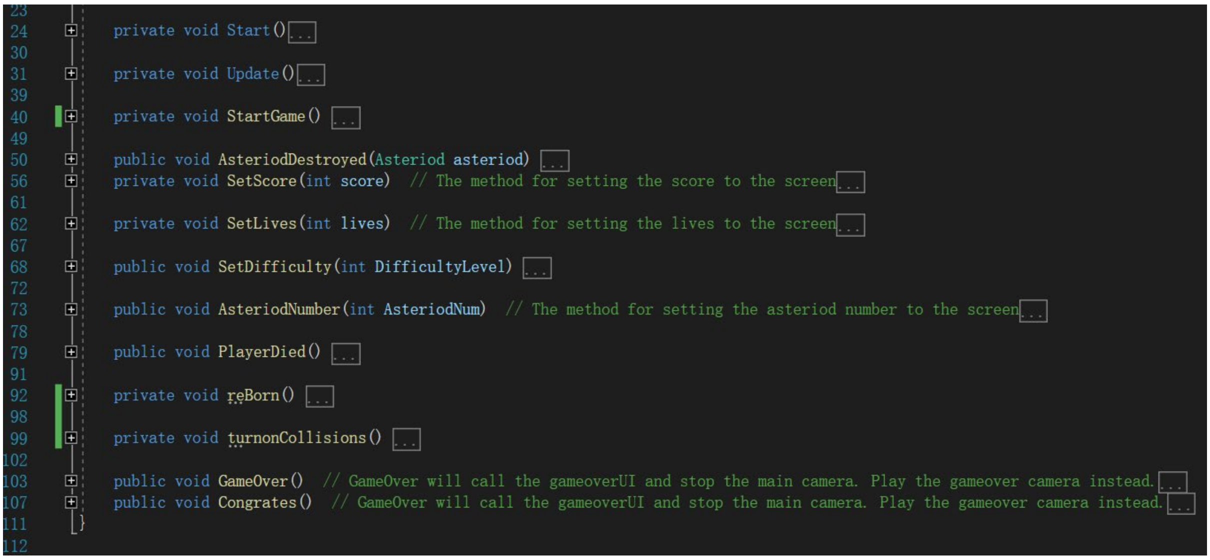
In introduction part, I said that apart from AI, I add a new function. This method is written because I found that sometimes the plane may reborn among a lot of asteroids, which may cause immediate collision after reborn and player may not have enough reaction time to escape from the crowds of asteroids. This method is updated in GameManager script and will temporarily turn off the collision detection between the plane and asteroids after reborn for 3 seconds, which allows player to react and escape. I use the layer change skill to achieve this method.



```
private void reBorn() {      // If live bigger than zero, than make the player reborn
    this.player.transform.position = Vector3.zero;  // Reborn position to the (0,0,0)
    this.player.gameObject.layer = LayerMask.NameToLayer("IgnoreCollisions");  // thrn off the collision detection, reasons are as mentioned as below next comment
    this.player.gameObject.SetActive(true);
    Invoke(nameof(turnonCollisions), 3.0f);            // After reborned, there are 3 seconds for player to react and escape in avoidance of immediate collision with asteroids after reborn
}

private void turnonCollisions() {          // thrn on the collision back again
    this.player.gameObject.layer = LayerMask.NameToLayer("Player");
}
```

Figure 3. Ignore Collision methods in GameManager script

The other part of GameManager keep unchanged as the old version.

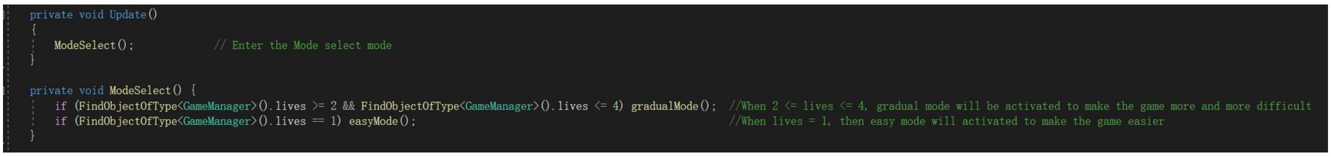


```
    private void Start()...

    private void Update()...

    private void StartGame() ...

    public void AsteriodDestroyed(Asteriod asteriod) ...
    private void SetScore(int score)  // The method for setting the score to the screen...

    private void SetLives(int lives)  // The method for setting the lives to the screen...

    public void SetDifficulty(int DifficultyLevel) ...

    public void AsteriodNumber(int AsteriodNum)  // The method for setting the asteriod number to the screen...

    public void PlayerDied() ...

    private void reBorn() ...

    private void turnonCollisions() ...

    public void GameOver()  // GameOver will call the gameoverUI and stop the main camera. Play the gameover camera instead....
    public void Congrates()  // GameOver will call the gameoverUI and stop the main camera. Play the gameover camera instead....
}
```

Figure 4. Skeleton of new version of GameManager script (107 lines)

The most important script in this assignment is the AsteroidAI script which is written based on the state machine model and controls all the AI behaviour of the asteroids.

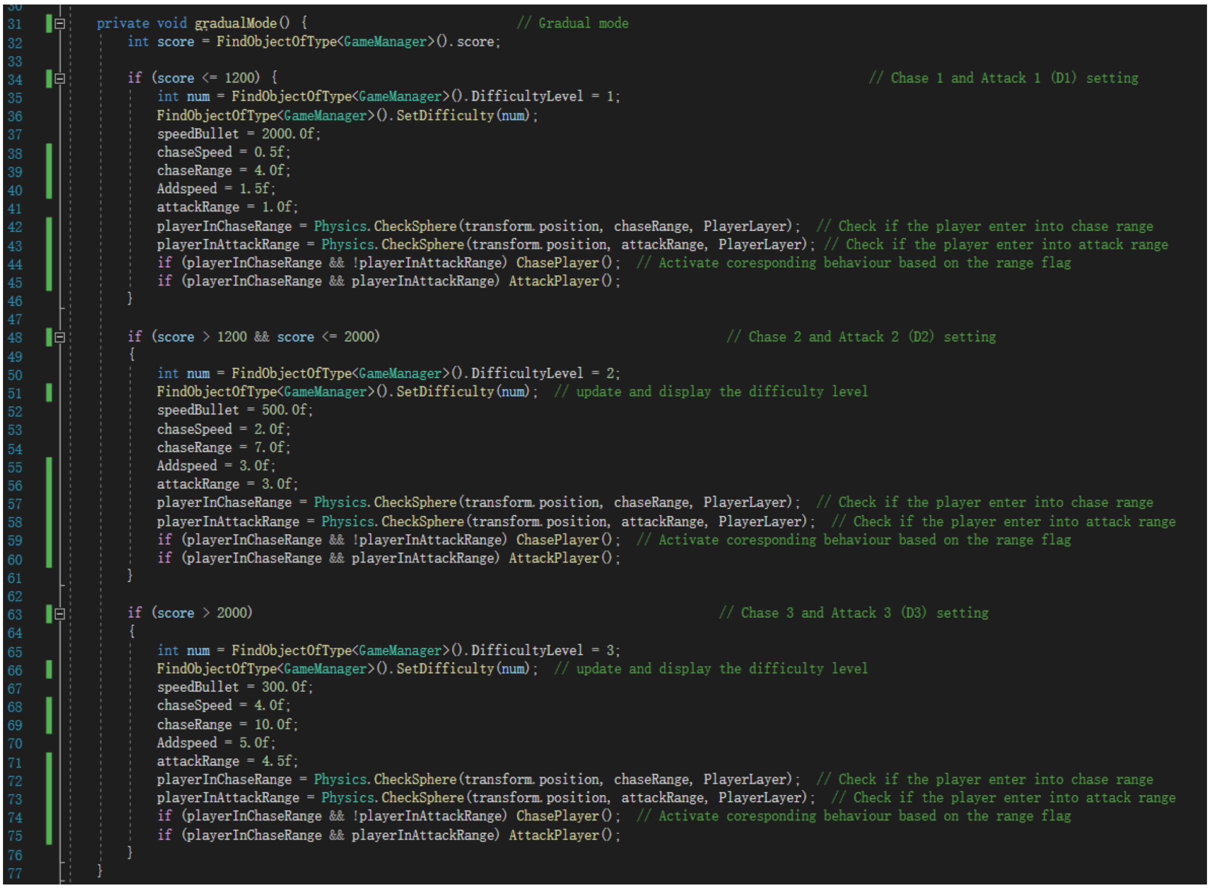
Based on the finite state machine, the program starts from the mode selection, so we need to select the overall mode:



```
private void Update()
{
    ModeSelect();             // Enter the Mode select mode
}

private void ModeSelect() {
    if (FindObjectOfType<GameManager>().lives >= 2 && FindObjectOfType<GameManager>().lives <= 4) gradualMode();  //When 2 <= lives <= 4, gradual mode will be activated to make the game more and more difficult
    if (FindObjectOfType<GameManager>().lives == 1) easyMode();                                                   //When lives = 1, then easy mode will activated to make the game easier
}
```

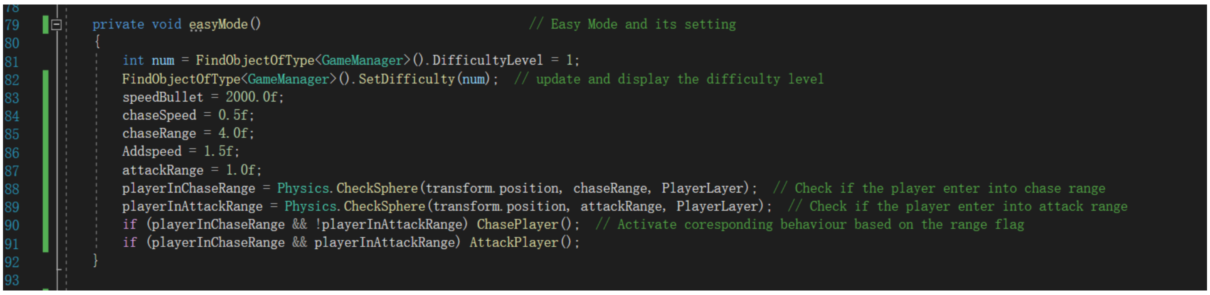
Figure 5. Mode selection

Then we need to implement the gradualMode and easyMode:



```csharp
private void gradualMode() {                                    // Gradual mode
    int score = FindObjectOfType<GameManager>().score;

    if (score <= 1200) {                                                                // Chase 1 and Attack 1 (D1) setting
        int num = FindObjectOfType<GameManager>().DifficultyLevel = 1;
        FindObjectOfType<GameManager>().SetDifficulty(num);
        speedBullet = 2000.0f;
        chaseSpeed = 0.5f;
        chaseRange = 4.0f;
        Addspeed = 1.5f;
        attackRange = 1.0f;
        playerInChaseRange = Physics.CheckSphere(transform.position, chaseRange, PlayerLayer);  // Check if the player enter into chase range
        playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, PlayerLayer); // Check if the player enter into attack range
        if (playerInChaseRange && !playerInAttackRange) ChasePlayer();  // Activate coresponding behaviour based on the range flag
        if (playerInChaseRange && playerInAttackRange) AttackPlayer();
    }

    if (score > 1200 && score <= 2000)                                          // Chase 2 and Attack 2 (D2) setting
    {
        int num = FindObjectOfType<GameManager>().DifficultyLevel = 2;
        FindObjectOfType<GameManager>().SetDifficulty(num);  // update and display the difficulty level
        speedBullet = 500.0f;
        chaseSpeed = 2.0f;
        chaseRange = 7.0f;
        Addspeed = 3.0f;
        attackRange = 3.0f;
        playerInChaseRange = Physics.CheckSphere(transform.position, chaseRange, PlayerLayer);  // Check if the player enter into chase range
        playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, PlayerLayer);  // Check if the player enter into attack range
        if (playerInChaseRange && !playerInAttackRange) ChasePlayer();  // Activate coresponding behaviour based on the range flag
        if (playerInChaseRange && playerInAttackRange) AttackPlayer();
    }

    if (score > 2000)                                                          // Chase 3 and Attack 3 (D3) setting
    {
        int num = FindObjectOfType<GameManager>().DifficultyLevel = 3;
        FindObjectOfType<GameManager>().SetDifficulty(num);  // update and display the difficulty level
        speedBullet = 300.0f;
        chaseSpeed = 4.0f;
        chaseRange = 10.0f;
        Addspeed = 5.0f;
        attackRange = 4.5f;
        playerInChaseRange = Physics.CheckSphere(transform.position, chaseRange, PlayerLayer);  // Check if the player enter into chase range
        playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, PlayerLayer);  // Check if the player enter into attack range
        if (playerInChaseRange && !playerInAttackRange) ChasePlayer();  // Activate coresponding behaviour based on the range flag
        if (playerInChaseRange && playerInAttackRange) AttackPlayer();
    }
}
```

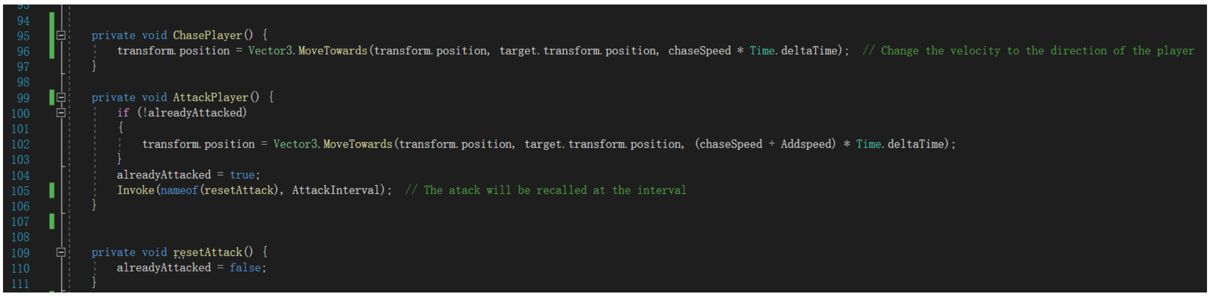
Figure 6. gradualMode



```csharp
private void easyMode()                                          // Easy Mode and its setting
{
    int num = FindObjectOfType<GameManager>().DifficultyLevel = 1;
    FindObjectOfType<GameManager>().SetDifficulty(num);  // update and display the difficulty level
    speedBullet = 2000.0f;
    chaseSpeed = 0.5f;
    chaseRange = 4.0f;
    Addspeed = 1.5f;
    attackRange = 1.0f;
    playerInChaseRange = Physics.CheckSphere(transform.position, chaseRange, PlayerLayer);  // Check if the player enter into chase range
    playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, PlayerLayer);  // Check if the player enter into attack range
    if (playerInChaseRange && !playerInAttackRange) ChasePlayer();  // Activate coresponding behaviour based on the range flag
    if (playerInChaseRange && playerInAttackRange) AttackPlayer();
}
```

Figure 7. easyMode

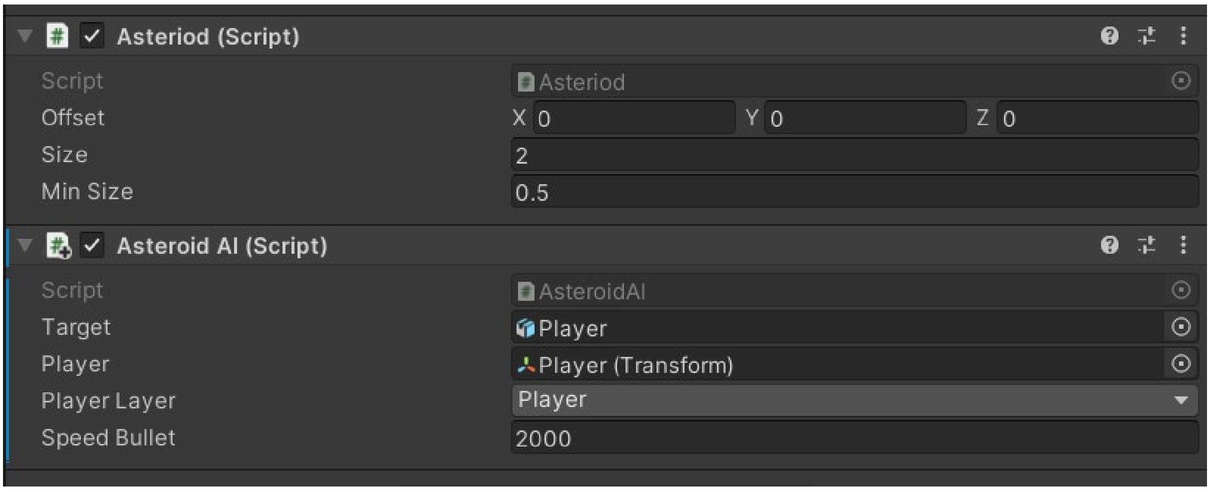
Among these two modes, the action ChasePlayer and AttackPlayer is defined as:



```csharp
private void ChasePlayer() {
    transform.position = Vector3.MoveTowards(transform.position, target.transform.position, chaseSpeed * Time.deltaTime);  // Change the velocity to the direction of the player
}

private void AttackPlayer() {
    if (!alreadyAttacked)
    {
        transform.position = Vector3.MoveTowards(transform.position, target.transform.position, (chaseSpeed + Addspeed) * Time.deltaTime);
    }
    alreadyAttacked = true;
    Invoke(nameof(resetAttack), AttackInterval);  // The atack will be recalled at the interval
}


private void resetAttack() {
    alreadyAttacked = false;
}
```

Figure 8. ChasePlayer and AttackPlayer

And each asteroid has two scripts loaded:



Asteriod (Script)

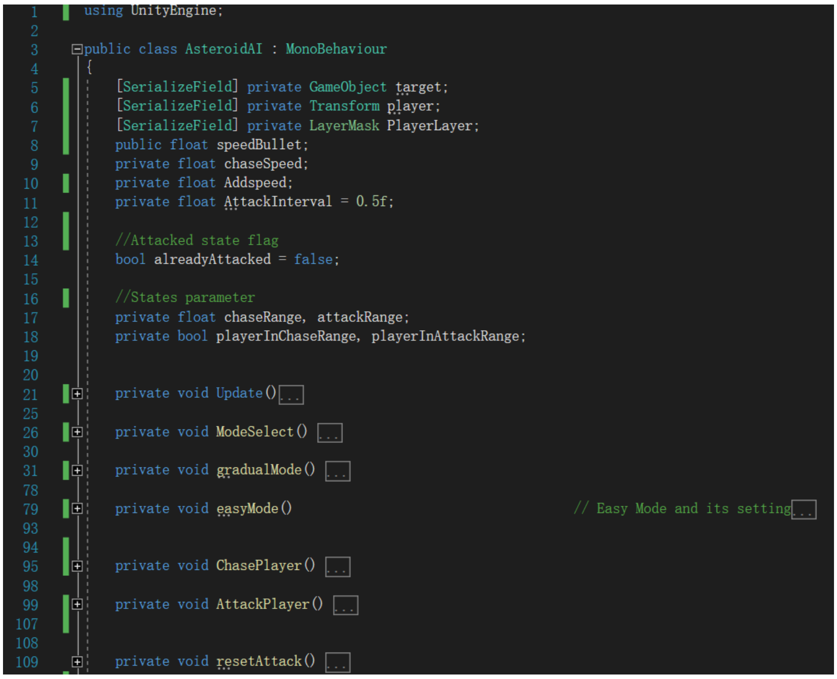
| | |
|---|---|
| Script | Asteriod |
| Offset | X 0 Y 0 Z 0 |
| Size | 2 |
| Min Size | 0.5 |

Asteroid AI (Script)

| | |
|---|---|
| Script | AsteroidAI |
| Target | Player |
| Player | Player (Transform) |
| Player Layer | Player |
| Speed Bullet | 2000 |

Figure 9. Two scripts loaded on the asteroids objects

So, when player is out of the chase and the attack range, the 'Asteroid AI' script will stop working and the first old script 'Asteroid' will come into main function, which is the 'Normal' state in the state machine. So, with the help of the two scripts: 'Asteroid' and 'Asteroid AI', the state machine model is well developed on the asteroid object.



```
using UnityEngine;

public class AsteroidAI : MonoBehaviour
{
    [SerializeField] private GameObject target;
    [SerializeField] private Transform player;
    [SerializeField] private LayerMask PlayerLayer;
    public float speedBullet;
    private float chaseSpeed;
    private float Addspeed;
    private float AttackInterval = 0.5f;

    //Attacked state flag
    bool alreadyAttacked = false;

    //States parameter
    private float chaseRange, attackRange;
    private bool playerInChaseRange, playerInAttackRange;


    private void Update()[...]

    private void ModeSelect() [...]

    private void gradualMode() [...]

    private void easyMode()                                   // Easy Mode and its setting[...]


    private void ChasePlayer() [...]

    private void AttackPlayer() [...]


    private void resetAttack() [...]
```

Figure 10. Skeleton of new script 'AsteroidAI' (109 lines)