Logo

Description automatically generated**Department of Applied Computer Games (DACG)**

**Games Programming 3 – MHI625659-21-A**

**Coursework 1**

**2021-2022**

**Submitted for the Degree of:**

**BSc Computer Games (Software Development)**

|  |  |
| --- | --- |
| **Student Name:** | Stephen Ross Cartner |
| **Matriculation Number:** | S1706321 |
| **Course:** | Games Programming 3 |
| **Word Count:** | 1223 |
| **Github Link:** | <https://github.com/Ubitroll/Games-Programming-3-Submission-Stephen-Cartner> |
| **Application Video:** | Youtube – <https://youtu.be/kq80DG78J-g>  Also Contained within Github files |

*I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award*.

*Signature*: Stephen Ross Cartner Date: 10/01/2022

**Contents**

1. Advanced Movement 3
   1. Orbit 3
2. Abstraction 5
   1. GameObject 5
      1. Vector Motion 5
      2. Orbit Function 5
   2. DeltaTime 5
3. Artificial Intelligence 7
4. Further Action and Continuation 8

**1.0 Advanced Movement**

1.1 Orbit

The orbit function works off a basic function that adjusts the forward-facing vector and the upwards-facing vector of a game object relative to a position that is input into it. This position (Shown in figure 1.1 as modelPos) is used as a fixed point for the GameObject to orbit around.

If the GameObject has moved to a new position, then it will have a new up vector between the GameObject and its target orbit point. A vector is first generated by normalizing the difference between the GameObject’s current position and the point it wishes to orbit. This generates a vector that points directly towards the orbit position however for the model position this acts as a down vector which must then flipped to give the new up vector of the GameObject. (See Figure 1.1) Text

Description automatically generated  
(Figure 1.1)

This new up vector is then used to determine the change in angle that the GameObject has undergone compared to its previous position. First the magnitude of the vectors is determined through the dot product of the GameObject’s old up vector and the new up vector (As shown in figure 1.2).

Text, letter

Description automatically generated

(Figure 1.2)

The magnitude is then divided by the square root of the product of each vector’s square being multiplied. This value being passed through the acos() function (Figure 1.3) will return a value between 0 and π radians which tells us the angle of rotation between the two desired vectors which is stored in a float.

Diagram

Description automatically generated

(Figure 1.3)

Now that the angle between the up and new up vector has been determined the forward-facing vector must be adjusted by this angle to make the new forward vector. Since the GameObject is rotating around it, the right vector won’t be changed due to the rotation so can be calculated using the previous up and forward vectors (See Figure 1.4). The new forward vector is then calculated by normalizing the old forward vector after it is multiplied by the rotation matrix (Figure 1.1). Finally, the up vector is set to equal the new up vector so that the next cycle of the game loop can begin the process again.

Diagram

Description automatically generated

(Figure 1.4)

What this process achieves is that every frame the GameObject’s up vector is adjusted to rotate it relative to the point it is orbiting around. The angle (shown in figure 1.4 as Θ) of difference is then applied to the forward-facing vector allowing its forward vector motion to move it around the desired point of orbit.

**2.0 Abstraction**

2.1 GameObject

2.1.1 Vector Movement

The Vector motion of the Camera system was repurposed within the GameObject class to abstract movement into simple functions like “moveForward” and “turnRight”. The functions were abstracted since all GameObjects need motion which compacts and simplifies the code for development. The function works by retrieving the GameObjects current position before moving it by the related directional vector. This coupled with the orbit function will ensure that all motion can be handled through methods (See Figure 2.1).

Text

Description automatically generated

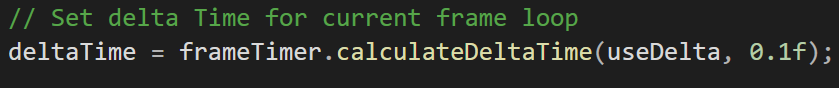
(Figure 2.1)

2.1.2 Orbit Function

The orbit function was abstracted into the GameObject class since this is a function that all the GameObjects would have to do besides the planet itself. This allows the asteroids, missiles, and the ship to rotate around the planet in an orbit (See Figure 1.1).

2.2 DeltaTime

Previously the delta time was calculated every time it was being used which for the size of program wouldn’t cause many issues. However, in a much larger program doing this would cause lag with many superfluous calculations so to optimise this the function was abstracted into its own class so that it could be called at the start of each game loop and used throughout it. This turns what could be thousands of important calculations into only one (See figure 2.2).



(Figure 2.2)

The delta time function works by dividing the time between frames by the frequency of the frames. This gives an incredibly small number which when used to multiply movement speed can smooth out the movement to match the frame rate of the program. The benefit of this is visibly smooth motion (See figure 2.3).

Text

Description automatically generated

(Figure 2.3)

**3.0 Artificial Intelligence**

3.1 Finite State Machine

The first step of setting up the Finite State Machine (FSM) is planning out a state machine diagram (See Figure 3.1). This diagram shows a simple 2 state FSM that functions by having conditions that allow the state to swap between the two functions.

Diagram

Description automatically generated

(Figure 3.1)

When the game starts the AI is started in the TRACK state. The purpose of this state is to point the AI towards the player and slowly move it towards them. It achieves this using the abstracted functions of the GameObject to set the forward vector of the AI ship to face towards the player before moving it along this vector. The function then calculates the distance between the AI ship and the player ship using their positions to begin the conditional statement. If the distance between the player and the AI ship has been reduced below a set threshold then the AI will be swapped into its ATTACK state (See Figure 3.2).

Text

Description automatically generated

(Figure 3.2)

When the FSM is in the ATTACK state it ceases its movement and instead turns into a stationary turret. Using a similar process as to how the camera can turn to face an object the Ship’s forward vector is set to follow the player. The AI then begins to fire missiles along this forward vector. Similar to the TRACK state the function determines the distance between the AI ship and the player before checking this against the pre-set value. If the AI is now too far away from the player, then the state is swapped back to TRACK so the AI may continue its pursuit (Appendix 3C).

A picture containing text

Description automatically generated

(Figure 3.3)

**4.0 Further action and Continuation**

Were I to continue this project or do something of a similar ilk later down the line there are a few things I’ve learned from this project that I would further implement into the program.

Firstly I would polish the movement capabilities of the GameObjects so that movement would be handled by forces and velocities which would not only provide smoother movement for the GameObjects but would also allow the AI system to make use of steering behaviours which apply forces to achieve desired movement patterns such as being apple to chase a target, predict where the target is going to be headed to fire at them or create a basic Seek function that would allow for missiles to home in on nearby targets.

I’d also choose to abstract the method for drawing the model into the GameObject class to further simplify the programming side of adding new GameObjects into the scene and also make it easier to control the orientation of the model mesh to match that of the objects forward and up vectors creating much more appealing visuals to the model movement.

The last thing I would like to implement is an abstraction of the AI class that would take in a specific GameObject to apply the AI’s movement and actions to. With this multiple AI Agents could be loaded into the same scene creating more enemies or challenges for the player to fight against whilst also reducing the complexity of the code for multiple AI significantly.