

|  |  |
| --- | --- |
| **Name** | William Tarrant |
| **T Number** | T00019474 |
| **Course** | Computing with Games Development |
| **Date Submitted** | 1 February 2013 |

# Project Description

This will be a **3D** game that will be designed for the web but which hopefully will also be ported to the android platform where the user will take control of a fighter craft and partake in missions where they must shoot down invading enemy fighter aircraft. It will be programmed using Unity and C# as the primary coding language.

The player will take a third person view and patrol the sky’s, looking for AI controlled enemy craft, the player must first align themselves with the enemy craft before they obtain missile lock; this applies also for the enemy craft. Once the missile is launched, it will fall under its own decision making controls and use captured data on the targets position in order to determine its velocity so that it can track, outsmart and calculate an intercept on its target (detailed below). The missile must obey the laws of physics to a certain extent in that its flight must be realistic (no sharp turns). The enemy craft will have the ability to drop flairs as a decoy the missile to avoid hits. The above rules will also apply for the players craft and will also have access to a number of flairs.

As I am aiming for a military type simulation for this project the main algorithm here will involve the missile guidance system and a secondary algorithm that will control enemy aircraft through an AI system. As mentioned, getting missile lock means line of sight (or falling within a cone of sight); this must be done to obtain a preliminary heat signature of the target. The AI will operate under an algorithm that will track the player’s movements using tools like radar and then analyse this data being obtained and try and make a best decision in obtaining its goal. Because we are dealing with two objects moving at high speeds and while also taking into account possible erratic manoeuvres the missile guidance system must continuously track the position of the plane, store and analysis the results and calculate a best path to intercept its target. Certain considerations must be taken into account when plotting an intercept course such as – Is it feasible, does the missile have the speed and fuel available to make this chosen path viable? This I am hoping should make gameplay more interesting as it is simply not a case of one object chasing the targets path but comes down to a game of psychology as here is an object making its own decision on how it is going to intercept you as a player base on your actions.

# Flow of Events (Missile)

Missile Path

Player / AI

Target

Launch – Fire and Forget Air to Air Missile

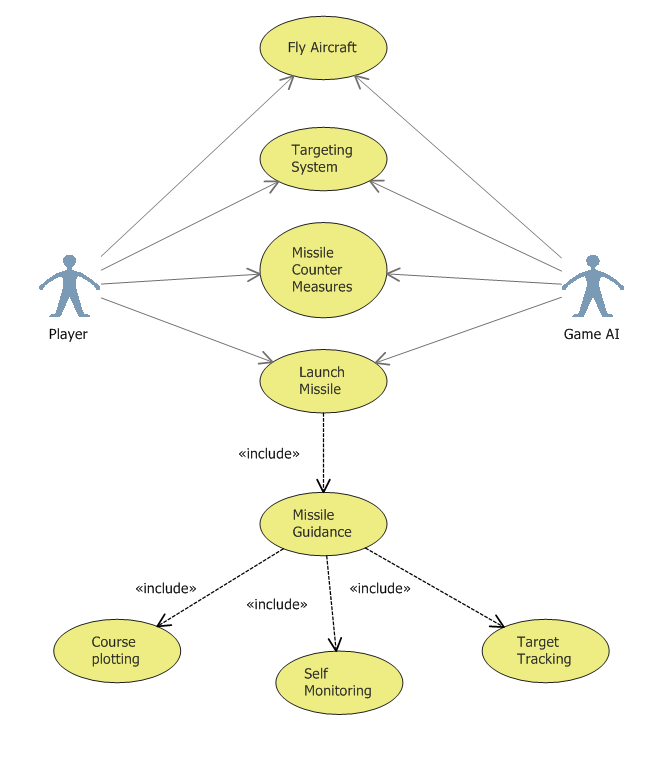
Radar ping

Predicted intercept based on analysis of captured data

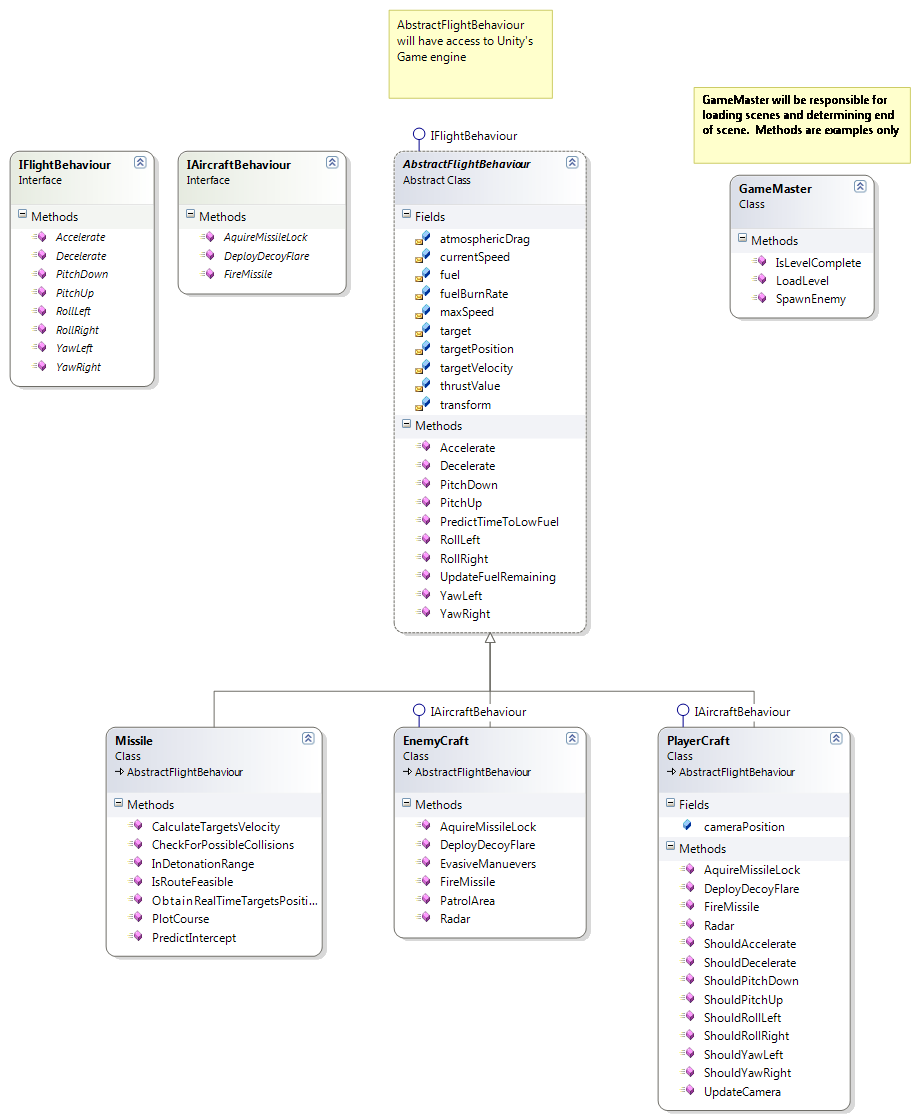
Missile alters intercept path



# Use Case Diagram



# Provisional Class Diagram



# Algorithm pseudo code

// Variables

:OldTargetPostion: Vect3

:CurrentTargetPos: Vect3

:TargetsVelocity: Vect3

:TargetsSpeed: float

:FlightPath: List<Vect 3>

// Methods

**Update()** //update method fires approx. every 1/60th of a second

currentTargetPos = **ObtainRealTimeTargetsPosition()**

TargetVelocity, TargetsSpeed = **CalculateTargetsVelocity(**currentTargetPos, OldTargetPosition, gameTime**)**

**CollisionAvoidance()**

**CheckIfInDetonationRange()**

Flight.add(TargetsVelocity, TargetsSpeed)

**PredictIntercept()**

**CheckForPossibleCollisions()**

If **IsRouteFeasible()** then // indicates route is feasible then

**PlotCourse()**

E**lse:**

Continue

**CalculateTargetsVelocity(vect3 Curr, Vect3 Old, float gameTimeElapsed)**

Calculate targets velocity based on old and new position vector proportional to the time measured between radar pings

Return: Speed and target velocity

**PredictIntercept()**

If target is travelling in a linear path then

Just edge closer

Else:

Based on targets velocity calculate where target will be in X time

Based on previous measured behaviour factor this in to predicted intercept point

Return: Path to intercept

**IsRouteFeasible()**

Based on fuel consumption, current fuel load calculate fuel cost and determine if route is feasible

Return: true or false

**CheckInDetonationRange()**

If within range then

Explode //End of Missile life

Trigger Event //inform subscriber of detonation. Allow for clean-up etc.…

**Else**

Continue

**PlotCourse()**

Alter pitch/yaw to point in direction of new intercept point