Name	William Tarrant
T Number	########
Course	Computing with Games Development
Module	Computing Project 2
Date Submitted	26 April 2013

Statement of originality

I, William Tarrant student number #########, the author, whereby certify that to the best of my knowledge that the material submitted for this project is the result of my own work except where stated explicitly otherwise and in which case the source of the material will be provided.

_____ _____

William Tarrant **Date**

Acknowledgement of works used in accomplishing of this project

Detonator Explosion Framework

This pack is used for game explosions which is available and is free to use from the unity asset store by the Unity group.

3D Models

The apache helicopter model was obtained from Blend swap under a Creative Commons Zero license. The model was uploaded by user "KuhnIndustries" and is available at http://www.blendswap.com/blends/view/44235

The AIM-9 missile model was obtained from Blend swap under a Creative Commons Zero license. The model was uploaded by user "TwoLpGamer" and is available at http://www.blendswap.com/blends/view/22736

Contents

Statement of originality 2
Acknowledgement of works used in accomplishing of this project 2
Detonator Explosion Framework 2
3D Models 2
Introduction 4
What is this game about? 4
How to play 4
Controls 4
Game feature description 5
The Missile Intercept 5
Radar system 8
The Multiplayer 9
Issues faced during the project 11
Unit testing 11
Version Control 11
Integrated Development Environment 12
Multiplayer Networking 13
Class Diagram 14

Introduction

What is this game about?

This is a multiplayer first person tactical shooter that involves flying an apache helicopter over a desert island with the simple objective being to shoot down your opponent before they shoot you down. The game has been built in Unity3D 4.1.2 with the main focus being given to providing an algorithm that controls a missile from launch to terminal guidance. For a missile to be efficient, it will normally try and always lead its target rather than just chase it.

The co algorithm used is the radar system that both the aircrafts and missiles have. This is instrumental for the missile to work as it facilitates the detection and monitoring of targets for determining their current position and velocity.

The game is also designed to be multiplayer with the choice of playing a game over the internet or the Local Area Network with up to 4 players.

How to play

1. Double click the RedSKY executable.
2. Enter a player name.
3. Decide whether to:
    a. Host a game through the internet
    b. Search and Join a game in progress
    c. Host a game on the Local Area Network
    d. Search and Join a game in progress

Controls

W Accelerate

A Yaw left

S Reverse

D Yaw Right

Q Pitch Up

E Pitch Down

X Roll Right

Z Roll Left

F Fire

Tab Toggle target

Game feature description

The Missile Intercept

For normal sidewinder missiles, the method for target tracking is done with an infrared sensor mounted on a gimbal and use of proportional navigation where if two moving objects remain on a constant bearing and appear to be getting closer then they will collide. [Source http://en.wikipedia.org/wiki/Infrared_homing] [Source http://en.wikipedia.org/wiki/Proportional_navigation]

The approach I wanted to take with my algorithm was to use radar instead of infrared. The radar system employed is the same as that used with the aircraft in my game and is described later on. Researching a method for target interception was initially difficult as a lot of the information available was heavily mathematical and thesis based and well beyond anything I wanted to implement at this stage.
I did in the end come across an excellently described method in a blog posting that allowed the predicting of an interception in 2D based on targets known velocity and a missile with a known top speed. The method does not account for acceleration but assumes an instantaneous and constant velocity.

[Source: http://jaran.de/goodbits/2011/07/17/calculating-an-intercept-course-to-a-target-with-constant-direction-and-velocity-in-a-2-dimensional-plane/#respond]

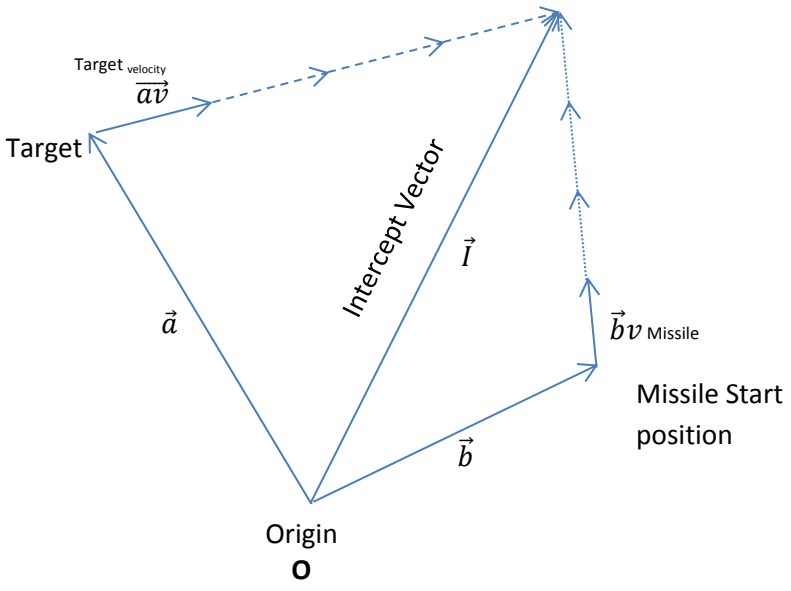


Figure 1 Calculation of intercept vector

The theory works based on the fact that the Intercept vector is common for both the missile and the target.

$\mathrm{I}_{\text{target}} = \vec{a} + t.\overrightarrow{av}$

$\mathrm{I}_{\text{missile}} = \vec{b} + t.\overrightarrow{bv}$

What we know

- Targets position can be discovered
- Targets velocity can be measured
- The missiles position is known
- The length of the missiles velocity vector

What is not known is the **time** but can be discovered because:

$$\mathrm{I}_{\text{target}} = \mathrm{I}_{\text{missile}}$$

So

$$\text{Target} + (\text{time} * \text{Target}_{\text{velocity}}) = \text{Missile} + (\text{time} * \text{Missile}_{\text{velocity}})$$

As described in the blog post it would be difficult to solve for the above so the approach taken instead would be to use the formula $speed = \frac{distance}{time}$

Where s = Length of the velocity vector of the missile (max speed)

t = time.

d = distance of the vector from **Missile base** to the intercept $\mathbf{I}$

So $d = \overrightarrow{bI} = \vec{I} - \vec{b}$

Where $\vec{I} = \vec{a} + t.\overrightarrow{av}$

Substitute for $\vec{I}$

$\overrightarrow{bI} = (\vec{a} + t.\overrightarrow{av}) - \vec{b}$

And now back to the original speed formula we now perform manipulation on it so that it looks like the following and we substitute in for the distance.

s.t = $|\ (\vec{a} + t.\overrightarrow{av}) - \vec{b}\ |$

For simplification purposes the author next let's $\left(\ \vec{a} - \vec{b}\ \right) = \vec{o}$ and now performs the distance formula on the LHS of the above formula and solves.

$$s.t = \sqrt{(\vec{o}_x + t.\vec{a}_{vx})^2 + (\vec{o}_y + t.\vec{a}_{vy})^2}$$

$$s^2.t^2 = (\vec{o}_x + t.\vec{a}_{vx})^2 + (\vec{o}_y + t.\vec{a}_{vy})^2$$

$$s^2.t^2 = \vec{o}_x{}^2 + 2.\vec{o}_x.t.\vec{a}_{vx} + t^2.\vec{a}_{vx}{}^2 + \vec{o}_y{}^2 + 2.\vec{o}_y.t.\vec{a}_{vy} + t^2.\vec{a}_{vy}{}^2$$

Carry the LHS over to the RHS

$$0 = \vec{o}_x{}^2 + 2.\vec{o}_x.t.\vec{a}_{vx} + t^2.\vec{a}_{vx}{}^2 + \vec{o}_y{}^2 + 2.\vec{o}_y.t.\vec{a}_{vy} + t^2.\vec{a}_{vy}{}^2 - s^2.t^2$$

Simplify

$$t^2\left(\vec{a}_{vx}{}^2 + \vec{a}_{vy}{}^2 - s^2\right) + 2t\left(\vec{o}_x.\vec{a}_{vx} + \vec{o}_y.\vec{a}_{vy}\right) + \vec{o}_x + \vec{o}_y = 0$$

The above now forms a special case and can be solved by quadratic equation

$$t1, t2 = \frac{-b \pm \sqrt{b^2 - ac}}{a}$$

Where

$a = \vec{a}_{vx}{}^2 + \vec{a}_{vy}{}^2 - s^2$

$b = \vec{o}_x.\vec{a}_{vx} + \vec{o}_y.\vec{a}_{vy}$

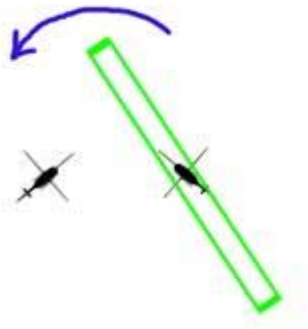
$c = \vec{o}_x + \vec{o}_y$

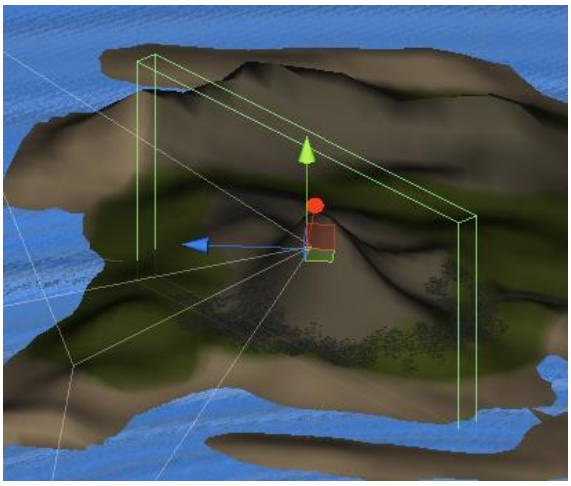
The lowest value of t will be the value wanted to substitute into this equation to calculate the Intercept vector.

$$\mathrm{I}_{\text{target}} = \vec{a} + t.\overrightarrow{av}$$

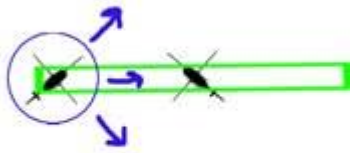
To convert the above to cater for 3d it is a simple case of tagging z on to a, b and c above.

Radar system

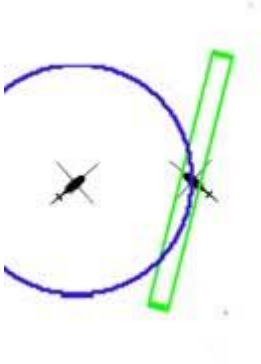


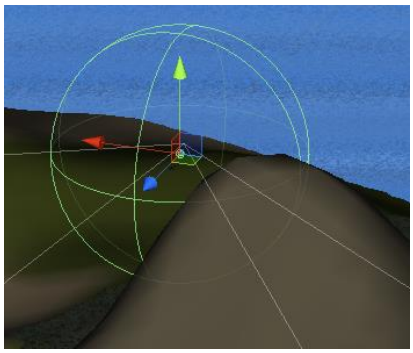


The player/owner craft sweeps the area with a bounding box. This is meant to represent a radar sweep of the area.



If the sweep collides with another craft, that craft spawns a bounding sphere that will continue to expand until it expires. This is to represent the ping back.





Once the owner/player craft receives the ping they begin to log the origin and signature of the received ping and from this the tracking process can begin.

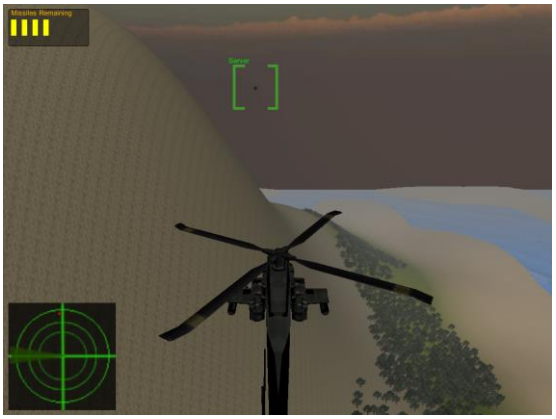


Figure 2 Players Radar HUD System

Once a target is being identified it is possible to monitor its position based on the last sweep. The velocity is calculated based on comparing the old cached position with the latest observed position from the radar sweep. This information is then used in the calculation of the predicted intercept.

The Multiplayer

Originally I had intended to use AI in the game to create enemy objectives. But having spent more time than intended getting the radar and missile algorithm in place I felt I wouldn't have had sufficient time to put in place a decent AI system.

As an alternative I decided to try using Unity's multiplayer features to create the enemy objectives.

Unity's Master Server

While Unity provided a full API for everything needed to use multiplayer. It was probably the hardest of all Unity's features to research on the internet. But came a across a good video that gave enough to get off the ground using Unity's free to use master server to register and discover games. The master server is a remote server run by Unity for testing that acts as a database for storing game names, IP addresses and ports so that they can be easily discovered. Unity uses a server / client approach to multiplayer where a game will register themselves as a server and all other games will join that server as a client. From following the video tutorial I didn't change much with how tutorial demonstrated how to register or obtained hosts as it worked - no need to change it. [Source http://cgcookie.com/unity/2011/12/20/introduction-to-networking-in-unity/]

The real work came about handling the many players and working out where certain logic takes place and doesn't need to be propagated across the network taking up bandwidth and processing time. An example of this is where does the radar sweep take place? In reality it happens simultaneously for both the player and the target but in a network simulation it only needs to take place on the local machine of the player. Having it happen on two machines at the same time is wasteful and could lead to conflicts in information similar to what can happen when using threads.

LAN game discovery

While it was great to be able to access the master server for registering games I wasn't satisfied with this being the only approach to register and discover games mainly because it relies on internet connectivity and firewall access.

With Unity's networking API it is very easy to just plug in an IP address and port number and create a connection. But the issue is how to discover what game is available and what the IP is and port number for the server hosting that game.

To do this I followed a simple tutorial on UDP multicast where it demonstrated two applications where one was a listener and the other was the sender and it would simply send a list of numbers and the listener would read and decode the messages. UDP was perfect in this case for my game as it is connectionless. [Source http://www.jarloo.com/c-udp-multicasting-tutorial/]

To make it work in the game I had to employ threading as I needed a listening process to run concurrently with the game. I needed to set up a listener that kicks off when the LAN server starts that will listen for any game that starts, looking to join. If they receive a game request message they will then multicast their IP and port number so that the requester can decide to join or not. On the client side it also needs a threaded listener that will be able to receive the message sent from the LAN server containing the IP and port number.

Issues faced during the project

Unit testing

One of the biggest issues and disappointments faced surrounded the fact that unity classes that inherit from monobehaviour cannot be instantiated as the “new” keyword is not supported. This made testing difficult and in some cases impossible. A solution I proposed and tried was to abstract as much behaviours and states and put them in a concrete class which my base monobehaviour class could then instantiate. Having a concrete class meant that it would be possible to create unit tests. Unfortunately a lot of features used in my application made use of monobehaviour and put them out of the scope of being tested completely. Examples being the networking and Radar systems developed in the application.

As I will detail later my woes with MonoDevelop, I have instead used Visual Studio 2012 and NUnit (version 2.6.2.12296). With Visual Studio and the extension manager I was able to obtain a package that runs the tests. The package I used for VS2012 was NUnit Test Adapter (Beta 4) which is a free extension.

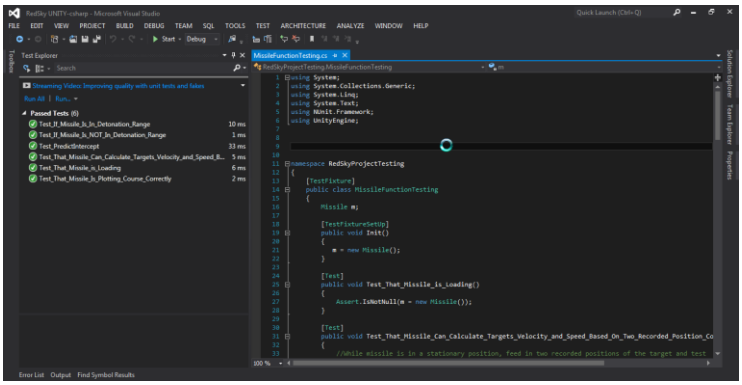


Figure 3 Running NUnit tests in Visual Studio 2012

A note of interest when using Unity3D, is that once any change takes place a new project is generated. For this purpose it is necessary to have the NUnit framework dll instead in a folder so that it can be accessed.

Version Control

Unity has made the use version control tricky when it comes to their indie package making it very difficult to collaborating on assets. The issue surrounds the fact that all assets are stored as binaries so when it comes to version control it is impossible for the VC system to distinguish who made what changes and will only overwrite the previous change. With careful management it is possible to work together having specific rules on who can work on what, when.

Unity pro is required to access the “force to text” feature so full version control is possible. However as a compromise for the indie package they introduced meta files that are stored alongside each file allowing the version control system know that are changes.

For my project I chose to use GIT for version control and GitHub for my offsite storage server. I went for GIT over the others for no particular reason other than I was curious to try it out. What I liked about it over the others is how it creates snap shots rather than versioning individual files. Also I like the fact it is distributed meaning it possible to work locally and commit changes without the need for

internet/network connectivity meaning that commits could be later pushed to the server. Another huge feature around GIT is that it is possible in the event of a failure be it the local machine or the server that it is possible to completely restore the repository based on which ever repository is still intact.

Something to note in creating a git repository for a Unity project is the “ignore” file (basically which files are not to be versioned). The following is a sample from my ignore file:

```
[Ll]ibrary/
[Tt]emp/
[Oo]bj/
Standard Assets/
```

The first three are the most important and should not be versioned as the files contained within can cause all sort of conflicts if working from different machines, this was an issue I encountered recently when trying to use DropBox for a different project and because these files were present and the files updated out of sync meant that one of my assets completely disappeared even though it was working perfectly when I saved and closed the application.

Regardless these files will be regenerated on whatever machine the project will be opened on again when checked out.

I decided not to check in any standard assets as they are common to Unity so it is not necessary to be checking in these all the time and waste bandwidth and storage.

Integrated Development Environment

While Monodevelop is a necessity for debugging, the version used by unity is out of date and is prone to crashes and its intellisense leaves a lot to be desired at times. Instead, out of pure frustration, for the majority of the time I resorted to using Visual Studio 2012 and used debug statements for discovering root of problems, not ideal in general but not having many options when it comes to use Monodevelop makes it ideal.

As mentioned previously, Unity, upon any change made will generate a new project meaning that any imported projects or references will be wiped. Originally, I had intended to have separate projects for the likes of game logic and testing but it became apparent very quickly that this was not going to be possible unless on every load, the projects need to be re-imported.

Multiplayer Networking

Any issue I encountered in the development of the multiplayer game through the use of Unity's master server was that for no apparent reason, 10 – 20 seconds in to the game, a drop of ~ 40 frames per second could be observed through the game statistics panel.

Initially, I felt it was a fault in my source code, but nothing obvious was showing. Later, upon searching the Unity support forums I discovered a post the resembled the exact same scenario that I was experiencing and it seemed to be a bug with Unity that resolves around the "Register Host" method when using the master server. I can confirm that this issue does not occur when using the LAN feature of the game.

When developing the LAN feature on my home network using multicasting, I discovered an issue where my multicast messages were not being broadcast, in the end the issue was tracked down to having Oracles Virtual Box installed and fact that one of its network adapters prevents multicast traffic. [Source https://www.virtualbox.org/ticket/8698]

Class Diagram