James Moran

This report shows certain parts of the development process for the Seek-and-Collect DirectX 11 application.

James Moran CGP600 AE2 – Individual Project Report

Contents

[Introduction 3](#_Toc503203561)

[Seek-and-Collect Overview 3](#_Toc503203562)

[Seek-and-Collect Features 3](#_Toc503203563)

[Initial Development Actions 3](#_Toc503203564)

[Additions to Satisfy the Basic Requirements 4](#_Toc503203565)

[Requirement 2 Implementation 4](#_Toc503203566)

[Requirement 3 Implementation 4](#_Toc503203567)

[Requirement 4 Implementation 4](#_Toc503203568)

[Requirement 5 Implementation 5](#_Toc503203569)

[Requirement 6/7/8/9 Implementation 5](#_Toc503203570)

[Enhancements 6](#_Toc503203571)

[Scene-Manager 6](#_Toc503203572)

[Project Conclusion 6](#_Toc503203573)

[Appendix A: Figures 8](#_Toc503203574)

[Appendix B: Class Hierarchy, Class Diagrams and Other Diagrams 16](#_Toc503203575)

[Class Hierarchy 16](#_Toc503203576)

[Enemy Finite State Machine Diagram 16](#_Toc503203577)

[Appendix C: Testing 17](#_Toc503203578)

[Feature Testing 17](#_Toc503203579)

[Evidence for Feature Testing 18](#_Toc503203580)

[Feature Test 1 18](#_Toc503203581)

[Feature Test 2 19](#_Toc503203582)

[Feature Test 3 20](#_Toc503203583)

[Feature Test 4 21](#_Toc503203584)

[Feature Test 5 22](#_Toc503203585)

[Feature Test 6 23](#_Toc503203586)

[Feature Test 7 25](#_Toc503203587)

[Feature Test 8 26](#_Toc503203588)

[Feature Test 9 26](#_Toc503203589)

[Feature Test 10 26](#_Toc503203590)

[Basic Testing 27](#_Toc503203591)

[Evidence for Basic Testing 29](#_Toc503203592)

[Basic Test 1 29](#_Toc503203593)

[Basic Test 2 29](#_Toc503203594)

[Basic Test 3 29](#_Toc503203595)

[Basic Test 4 29](#_Toc503203596)

[Basic Test 5 29](#_Toc503203597)

[Basic Test 6 29](#_Toc503203598)

[Basic Test 7 29](#_Toc503203599)

[Basic Test 8 29](#_Toc503203600)

[References 30](#_Toc503203601)

# Introduction

## Seek-and-Collect Overview

In Seek-and-Collect, you are the pilot of a hover-tank, delegated with the task of collecting Energy- Capsules. You are to push aside or avoid obstacles in your way whilst you go about collecting these capsules. You are also to watch out for any ‘Black Goons’ vehicles you see, who will attempt to destroy your hover-tank by ramming into it with their hover-tanks of a similar appearance (but with black-metal plating instead of the metallic-colour of your hover-tank’s plating).

## Seek-and-Collect Features

* The Player is to be represented by a static-mesh (as shown in Fig. 4 of Appendix A), that can move forwards, backwards, leftwards and rightwards, whilst also being able to yaw their hover-tank leftwards or rightwards (by 45° for each key-press)
* The Player and Enemies are not able to go through Static-Rocks (as represented by the static-mesh shown in Fig. 14 of Appendix A), stopping outside the bounding-sphere cast around them
* The Player and Enemies can push Wooden-Barrels (as represented by the static-mesh shown in Fig. 15 of Appendix A), moving the barrels as per their direction of movement into them
* The Player can pick-up Energy-Capsules (as represented by the static-mesh shown in Fig. 16 of Appendix A), by moving over them (removing them from the game-scene). Whilst Enemies are to simply pass through them (not removing them from the game-scene). If the Player picks-up all Energy Capsules in the game-scene, they will win that level, causing the game to freeze for 3 seconds, after which, it is terminated
* Enemies are to attempt to move towards the Player (following a bee-line), to deal damage to their hover-tank by ramming it with their own (as represented by the static-mesh shown in Fig. 17 of Appendix A). If they ram the Player 10 times, they will cause the Player to lose, removing the Player’s representation for their perspective, before sleeping for 3 seconds, then terminating the application

## Initial Development Actions

I started off by using an older Tutorial project (Tutorial 08 Exercise 01), as a basis for this assignment’s project. This is because I deemed this Tutorial project the most stable version of the project (with no discernible memory leaks). I went through the code, refactoring suitable functions/methods, as well as properties, into classes, with respects to which actions these methods/functions execute and which accepted answer (with the second highest number of up-votes), which simply involves adding $(WindowsSDK\_IncludePath) to the Include Directories section of VC++ Directories. This resolved the occurrence of these warning messages. (gradbot, 2012)

After resolving the issue noted in the above paragraph, I resolved an issue with DirectX Memory-Leaks, which I was made aware of, by D3D11 issuing warnings to me, via the output window. These appear after terminating the application, with messages like those listed in Appendix A: Fig. 1.

I resolved live objects not being cleaned-up (which is what these messages indicate), by following all the steps of the tutorial, that is available from the same source as the quote that is noted, in Appendix A: Fig. 1. (Master Kenneth, 2014)

# Additions to Satisfy the Basic Requirements

After the initial development actions, came the process of adding to the project, to satisfy the basic requirements (listed on the assignment brief).

First off, came that of adding functionality to the project to load and draw .obj files (assets, exported from a 3D-Modeling package, such as Autodesk’s 3D Studio-Max).

After the above system had been implemented appropriately, came the production of a simple model to represent the Player’s hover-tank, using 3DS Max 2017. Appendix A: Fig. 2 is the reference image that was used to create a static-mesh, to represent the Player’s hover-tank. (Mike Celestino, 2015)

After going through the process to develop this static-mesh, I was then able to show it in the default scene of the game. Appendix A: Fig. 3 is an image of the Player’s hover-tank static-mesh side-by-side with the reference image on a plane (render-image produced by 3DSMax). Appendix A: Fig. 4 shows the Player’s hover-tank as one can see it in the default scene of the game.

## Requirement 2 Implementation

As the first requirement has already been satisfied (as this application is a 3D game, using Visual Studio as the IDE, with object-orientated C++ in combination with DirectX (11) for the project’s implementation), I have now considered this requirement, at this stage of the project’s development cycle.

I started off the implementation required for this requirement, by setting up movement for the Player, in line with their Y-Rotation (Yaw), so they always move in the direction they are facing. After attempting various methods to set-up, a third-person camera, that follows the Player (iedoc, 2015), I settled on a method for an ‘Arc-Camera’ that keeps distance with the Player and always faces them, no matter the direction the Player’s hover-tank is facing. (Allen Sherrod and Wendy Jones, 2012)

## Requirement 3 Implementation

With a third-person camera now moving with the Player as they move through the level, came that of fulfilling the third requirement. I started off by going about the implementation of a StaticObstacle class, for static-obstacles in the game scene.

After this, came putting together the static-mesh for the static-obstacles in Scene0 (the first scene of the game). I used the reference image available under Appendix A: Fig. 5, to create this obstacle’s static-mesh. (© 1970-2017 Analytical Scientific, LTD.)

These can be seen in the default game-scene, with the Player’s hover-tank, in Fig. 6. of Appendix A.

I then put together the static-mesh for moveable-obstacles in Scene0 (the default scene). I used the reference image available under Appendix A: Fig. 9, to aid in the creation of this moveable-obstacle’s static-mesh (for that of a wooden barrel). (Independent.co.uk, 2014)

Finally, I put together the static-mesh for Energy-Capsules (collectable-objects) in the default scene. I used the reference image noted under Fig. 11 of Appendix A, to aid in the creation of the Energy-Capsule’s static-mesh. (9to5Google.com, 2013)

## Requirement 4 Implementation

With static and mobile obstacles, as well as collectibles (Energy Capsules) now in place in the game scene came the implementation required to meet the fourth requirement. I decided to add textures to the environment-objects first, starting with the static-rocks.

I looked for and found an image to be used for the texture of the static-rocks, on Textures.com (formerly CGTextures.com). This can be found under Fig. 7 of Appendix A. (Copyright © 2005-2017, Textures.com)

After this, I found an image to be used for the texture of the Player’s hover-tank on Textures.com. This can be found under Fig. 8 of Appendix A. (Copyright © 2005-2017, Textures.com)

I next found an image to be used for the texture of the default moveable-objects (wooden barrels) once again, on Textures.com. This can be found under Fig. 10 of Appendix A. (Copyright © 2005-2017, Textures.com)

Finally, I found an image to be used for the texture of the Energy-Capsules (collectable-objects) yet again, on Textures.com. This can be found under Fig. 12 of Appendix A. (Copyright © 2005-2017, Textures.com)

## Requirement 5 Implementation

With basic obstacles in the scene, that have lighting, came the 5th basic-requirement. For this, I went about the implementation of a collision system for all GameObjects.

I implemented a basic bounding-sphere collision system, which would find the centre point of each object (for the centre point of the sphere), then find the furthest vertex from this centre point (which would become the radius of this sphere), that one could then use for collision checking using Pythagoras’s theorem (if the distance between the centre points of the spheres is less than that of the sum of each sphere’s radius, then a collision has occurred as per this system). This allows for static-obstacles (rocks) to not be moved if a controlled-object collides with them (such as the Player’s hover-tank), along with the moveable-obstacles (wooden barrels) to be pushed by the Player if they collide with them. Finally, this allows the Player to collect the energy-capsules (collectable-objects), within the level, so that the Player can win the current level. I used a method found online to handle removal of collectable-objects from the scene-objects collection. (Georg Fritzsche, 2010)

## Requirement 6/7/8/9 Implementation

This requirement requires the implementation of non-player entities into the game, I have implemented a class for such (EnemyHoverTank), which will use a similar static-mesh to the Player’s hover-tank (but edited slightly), along with a texture on top of such. I found a texture to distinguish Enemy hover-tanks from the Player’s at Textures.com. This can be found in Appendix A: Under Fig. 13. (Copyright © 2005-2017, Textures.com)

For the AI of the Enemy hover-tanks, a Finite State Machine (FSM) will be used (as they have simple behaviour, that can be aptly described in an FSM). This is available under Appendix B: Class Hierarchy, Class Diagrams and Other Diagrams: Enemy Finite State Machine Diagram.

The Player is also able to take damage from an enemy and if they take enough damage (from 20 collisions between them and the Enemy hover-tanks), they will lose. By extension though, the Player can win if they collect all the Energy capsules in the level (by default, there are 20 capsules to collect). Unfortunately, this is not explicitly mentioned to the Player (unable to set-up textures for a plane to show to the Player and/or texture to be drawn to the Player’s viewport).

# Enhancements

## Scene-Manager

After setting-up a third-person camera to follow the Player as they move, I went about the implementation of a GameScene class, for the SceneManager to manage the active scenes of the game (not only certain GameObjects), as per the design laid out for the SceneManager, by John McGrath. (John McGrath and James Moran, 2017)

The GameScene would then manage any GameObjects that are part of that scene, but only when it is active.

There is a global function (called in the scope of Main.cpp), that initialises the classes. When this function is called in the WinMain() function, assert() macro-calls are used to make sure the pointers to these classes, are valid. (cplusplus.com, 2000-2017)

For referring to pointers (not copying them), by certain classes, of which the pointer is that of another class, \*& is used for such (instead of \*\*), to call the functions of that pointer’s class. (Matthew Hoggan, 2017)

After refactoring all the components of the project into their own classes, came the process of cleaning-up the project, by resolving as many warnings as possible.

# Project Conclusion

In conclusion of this project, although a playable game has been produced (that one can win or lose at), I deem it not ready for release to any platform, as per the feature-set of Seek-and-Collect, that is detailed in the ‘Seek-and-Collect Features’ sub-section of the ‘Introduction’ section.

Looking back at the key systems and their implementation that stand out comes first, how window-resizing is handled.

For this, I consulted the Microsoft Development Network (MSDN) for two articles on window-resizing. One for a general overview of the process (Windows Dev Centre, 2018) specific documentation on the IDXGISwapChain::ResizeBuffers() method. (DXGI Documentation, 2018)

The next key system implementation (to remove Energy-Capsules from the scene, when the Player moves over them with their hover-tank), is the utilisation of the ‘Erase-Remove Idiom’. (Georg Fritzsche, 2010)

This was chosen as it makes sure that the respective item is removed from the vector completely (the vector handles resizing of itself, to account for this change). Evidence of this implementation working as expected, is shown in Feature Test 6 of the ‘Evidence for Feature Testing’ section of Appendix C: Testing.

Looking back at issues that occurred though, raises 2 issues that I encountered whilst completing the project: Texture-Mapping and Lighting.

On the issue of Texture-Mapping, although I was able to apply textures to all objects, the textures have not been applied to the object in the intended manner (either tiled or matching up appropriately to the object). I tried to apply textures to meshes in 3DSMax, before exporting the models with the textures to use in the project, but to no avail. For the future, I would want to consider how to properly set-up a texture map for a certain object, before exporting it from 3DSMax.

Regarding the issue of lighting for objects, although I have attempted to implement (at least) ambient lighting for objects, this would not show-up over the textures it seems. I also tried to export the lighting normal-directions from 3DSMax (so that lighting would be present when the objects are used in a game-scene), but once again, this still resulted in lighting not being discernible on objects. For the future, I would want to find out how to set-up lighting for models put together in 3DSMax properly, before exporting them from 3DSMax for use. If not that, I would want to make sure the lighting system is correctly set-up in a DirectX 11 project, for any .OBJ files used in it.

# Appendix A: Figures

Figure 1: A quote from masterkenth.com, as an example of D3D11 (simple) Warning messages:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | D3D11 WARNING: Process is terminating. Using simple reporting. Please call ReportLiveObjects() at runtime for standard reporting. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING: Live Producer at 0x007464B0, Refcount: 4. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x007476F8, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A07580, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A07264, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A10ACC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A11204, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A1295C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A11944, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A11B60, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A120C4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A133BC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A1373C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A14A2C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A14CB4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A153AC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A16D7C, Refcount: 1. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A22B7C, Refcount: 1. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x0C17154C, Refcount: 1. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A26904, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A2A2CC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x0C1687F4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x0C16BE0C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A1ADC4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING: Live                         Object :     22 [ STATE\_CREATION WARNING #0: UNKNOWN]  DXGI WARNING: Live Producer at 0x006AE3A8, Refcount: 4. [ STATE\_CREATION WARNING #0: ]  DXGI WARNING:  Live Object at 0x006AEA38, Refcount: 2. [ STATE\_CREATION WARNING #0: ]  DXGI WARNING: Live                         Object :      1 [ STATE\_CREATION WARNING #0: ] |

(Master Kennth, 2014)

Figure 2: A top-down view of a Lego interpretation of the Millennium Falcon, used as a reference image to aid in the product of the static-mesh to represent the Player’s hover-tank:



(Mike Celestino, 2015)

Figure 3: A top-down render-view from 3DSMax, of the Player’s hover-tank reference image shown side-by-side with the produced hover-tank’s static-mesh:

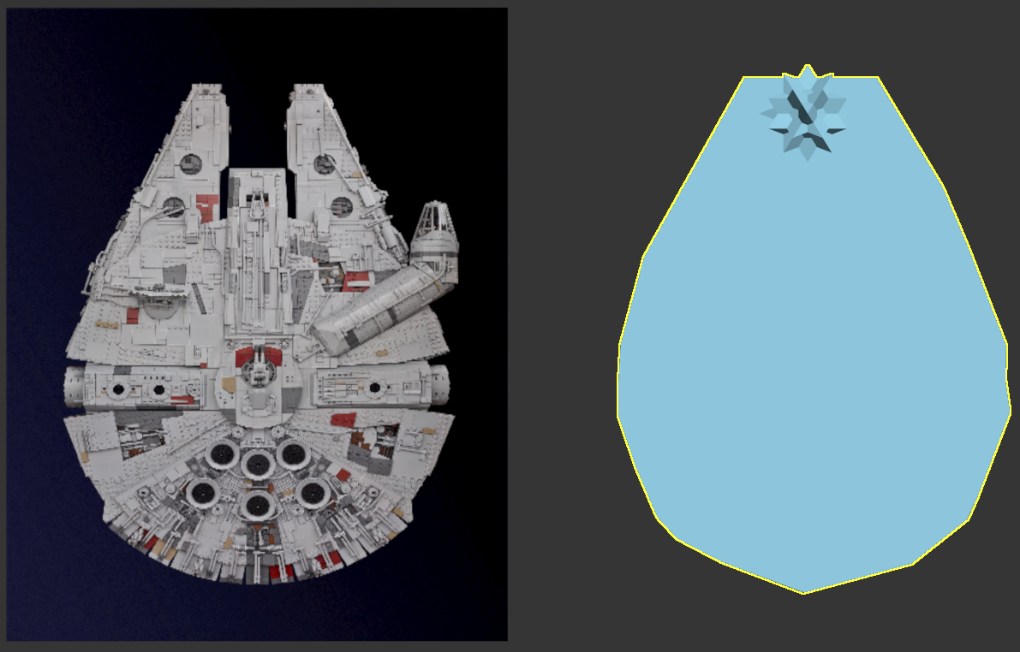


Figure 4: A perspective-view of the Player’s hover-tank’s static-mesh in the default game scene:

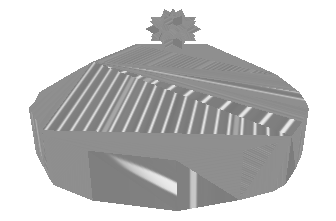


Figure 5: Side-on View of an igneous rock, used a reference image to produce static-obstacles in the default game scene:

(© 1970-2017 Analytical Scientific, LTD.)

Figure 6: The Player’s hover-tank with a few rocks, in the default game-scene:

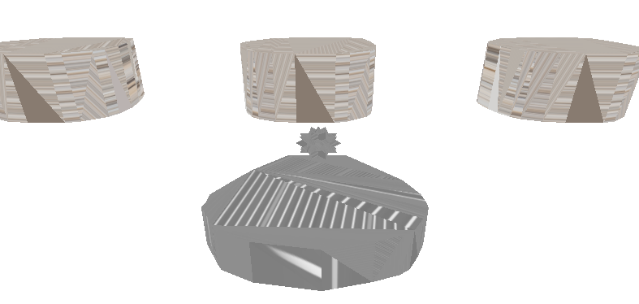


Figure 7: The image used to form a texture used for static-rocks in the default game-scene:

(Copyright © 2005-2017, Textures.com)

Figure 8: The image used to form a texture used for the Player’s hover-tank, to be applied to the Player’s hover-tank for all scenes of the game:



(Copyright © 2005-2017, Textures.com)

Figure 9: The reference image used to put together the default static-mesh to be used by moveable obstacles:



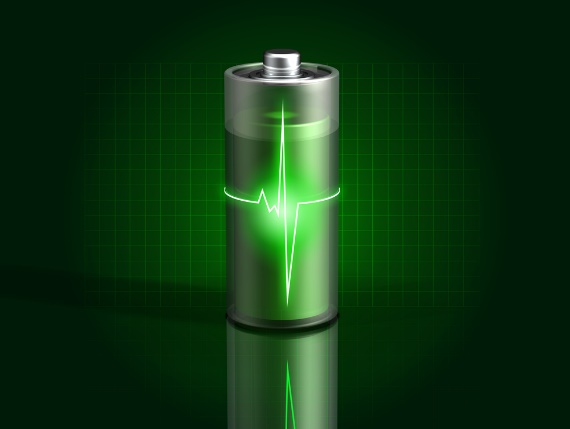
(Independent.co.uk, 2014)

Figure 10: The image used to form a texture to be used by the default moveable-obstacles (wooden barrels), in the default game-scene:



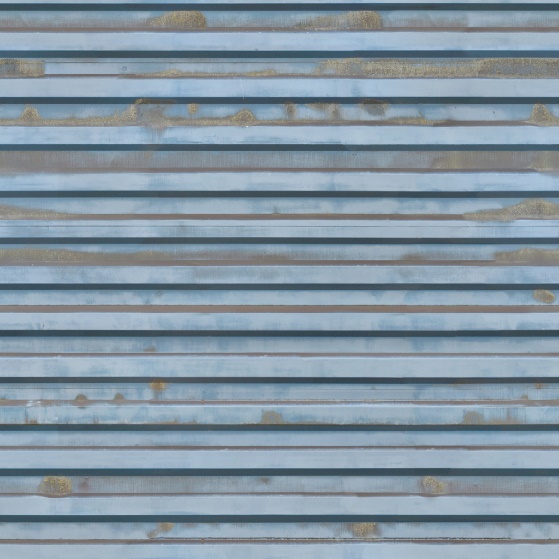
(Copyright © 2005-2017, Textures.com)

Figure 11: The image used a reference, for putting together the static-mesh to represent Energy Capsules (collectable-objects):

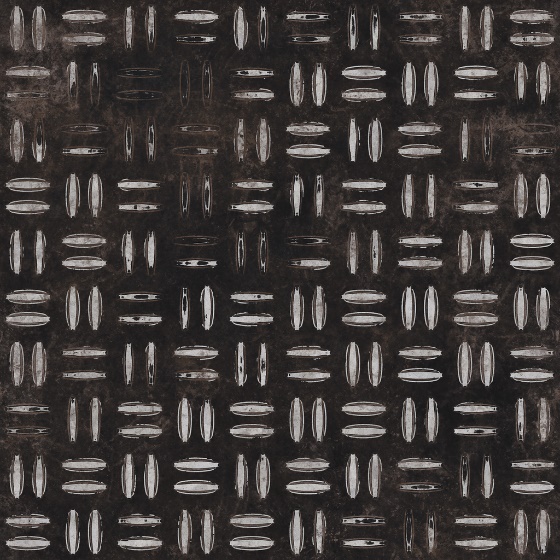


(9to5Google.com, 2013)

Figure 12: The image used to produce the Energy-Capsule’s default texture:



(Copyright © 2005-2017, Textures.com)

Figure 13: The image used to produce the Enemy hover-tank’s default texture:

(Copyright © 2005-2017, Textures.com)

Figure 14: Static-Rocks, immoveable by the Player or Enemies’ hover-tank:



Figure 15: Wooden-Barrels, moveable by the Player or Enemies’ hover-tank:



Figure 16: Energy-Capsules, the items that the Player is to collect to complete the level:

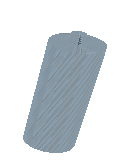
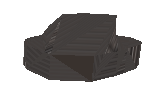


Figure 17: Enemy Hover-Tank Static-Mesh: To represent Enemy hover-tanks in the default game-scene:



# Appendix B: Class Hierarchy, Class Diagrams and Other Diagrams

## Class Hierarchy

The project’s class hierarchy:



## Enemy Finite State Machine Diagram

For the Enemy hover-tank’s AI:

# Appendix C: Testing

## Feature Testing

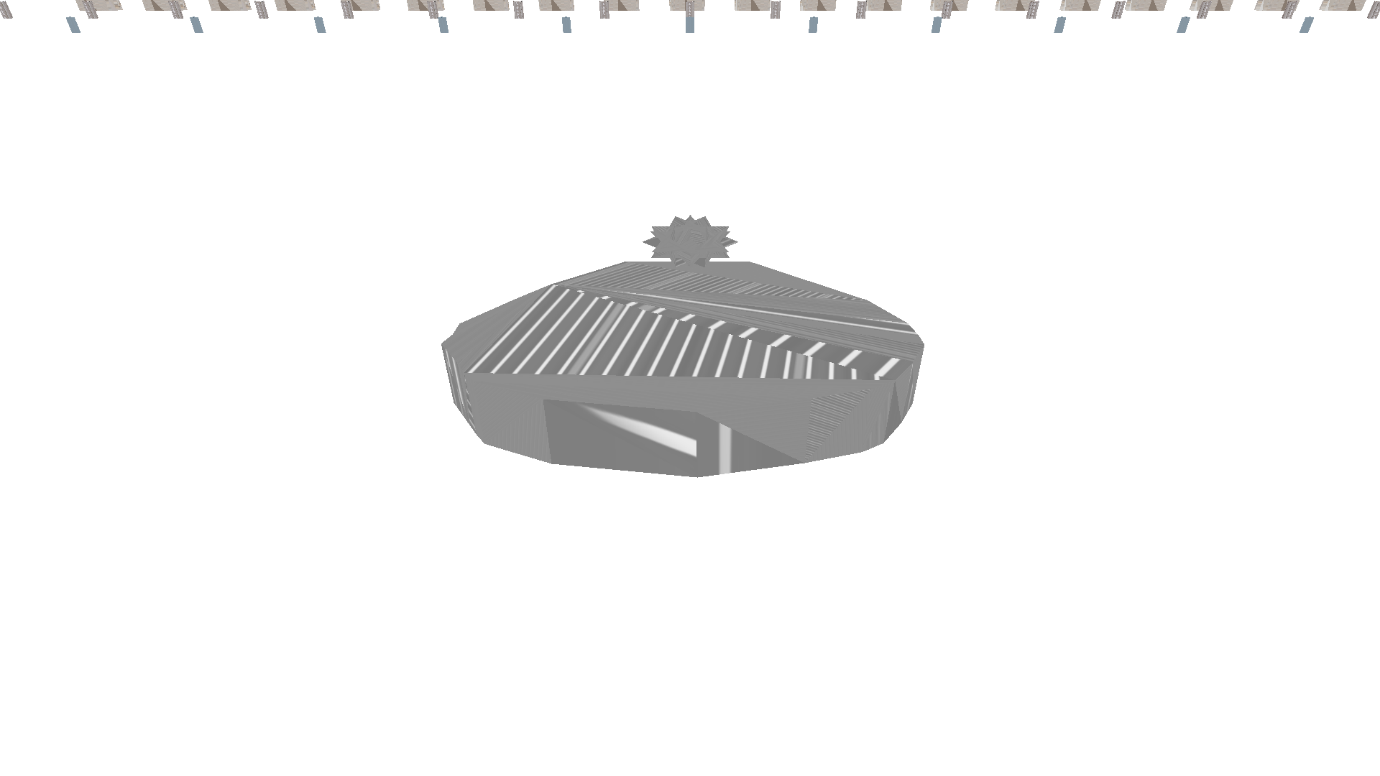
The tests of the features of Seek-and-Collect are documented here, as per the features outlined in the ‘Seek-and-Collect Features’ section of the introduction:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Test Description | Expected Result | Actual Result | Success or Failure |
| 1 | The Player’s static-mesh for their hover-tank is visible, as well as the other objects that are in their line-of-sight when the game is started. | The Player can see their hover-tank, as well as the Energy-Capsules, Wooden-Barrels and Static-Rocks of the level, when the game is initialised. | The Player can see their hover-tank, as well as the Energy-Capsules, Wooden-Barrels and Static-Rocks of the level, when the game is initialised. | Success. |
| 2 | The Player hover-tank’s movement is blocked by any Static-Rocks, disallowing their traversal through their bounds. | The Player’s hover-tank is not able to move through any given Static-Rock. | The Player’s hover-tank is not able to move through any given Static-Rock. | Success. |
| 3 | Enemy hover-tank’s movement is blocked by static-rocks, disallowing them to move through their bounds. | Enemy hover-tanks are not able to move through any given static-rock. | Enemy hover-tanks are not able to move through any given static-rock. | Success. |
| 4 | The Player can push Wooden-Barrels along with them, as per their movement direction. | Wooden-Barrels move along the Player hover-tank’s current direction of movement, when collided with. | Wooden-Barrels move along the Player hover-tank’s current direction of movement, when collided with. | Success. |
| 5 | Enemy hover-tanks can push Wooden-Barrels along with them, as per their movement direction. | Wooden-Barrels move along an Enemy hover-tank’s current direction of movement. | Wooden-Barrels move along an Enemy hover-tank’s current direction of movement. | Success. |
| 6 | The Player’s hover-tank can pick-up Energy-Capsules by colliding with them. When they collect all Energy-Capsules in the game-scene, they win the level. | The respective Energy-Capsule is removed from the game-scene when moved over by the Player.  When the Player has collected all Energy-Capsules, they win, causing the game to freeze for 3 seconds, before terminating. | The respective Energy-Capsule is removed from the game-scene when moved over by the Player.  When the Player has collected all Energy-Capsules, they win, causing the game to freeze for 3 seconds, before terminating. | Success. |
| 7 | Enemy hover-tanks are not able to pick-up Energy-Capsules by colliding with them. | The respective Energy Capsule is not removed from the game-scene when moved over by an Enemy hover-tank. | The respective Energy Capsule is not removed from the game-scene when moved over by an Enemy hover-tank. | Success. |
| 8 | Enemies are to attempt to seek out the Player, by moving towards them in a bee-line. | Enemies rotate to face the Player, before moving towards them in a bee-line, getting stopped by any Static-Rocks. | Enemies rotate to face the Player, before moving towards them in a bee-line, getting stopped by any Static-Rocks. | Success. |
| 9 | Enemies are to deal damage to the Player’s hover-tank by colliding with it. | After a collision, the Player’s hover-tank is dealt damage. | After a collision, the Player’s hover-tank is dealt damage | Success |
| 10 | If the Player receives 10 instances of collision from Enemy hover-tanks, they lose the game, with their hover-tank no longer being drawn to the game scene and the game freezing for 3 seconds before terminating. | The Player’s hover-tank’s static-mesh is no longer drawn and the game freezes for 3 seconds before terminating itself. | The Player’s hover-tank’s static-mesh is no longer drawn and the game freezes for 3 seconds before terminating itself. | Success |

## Evidence for Feature Testing

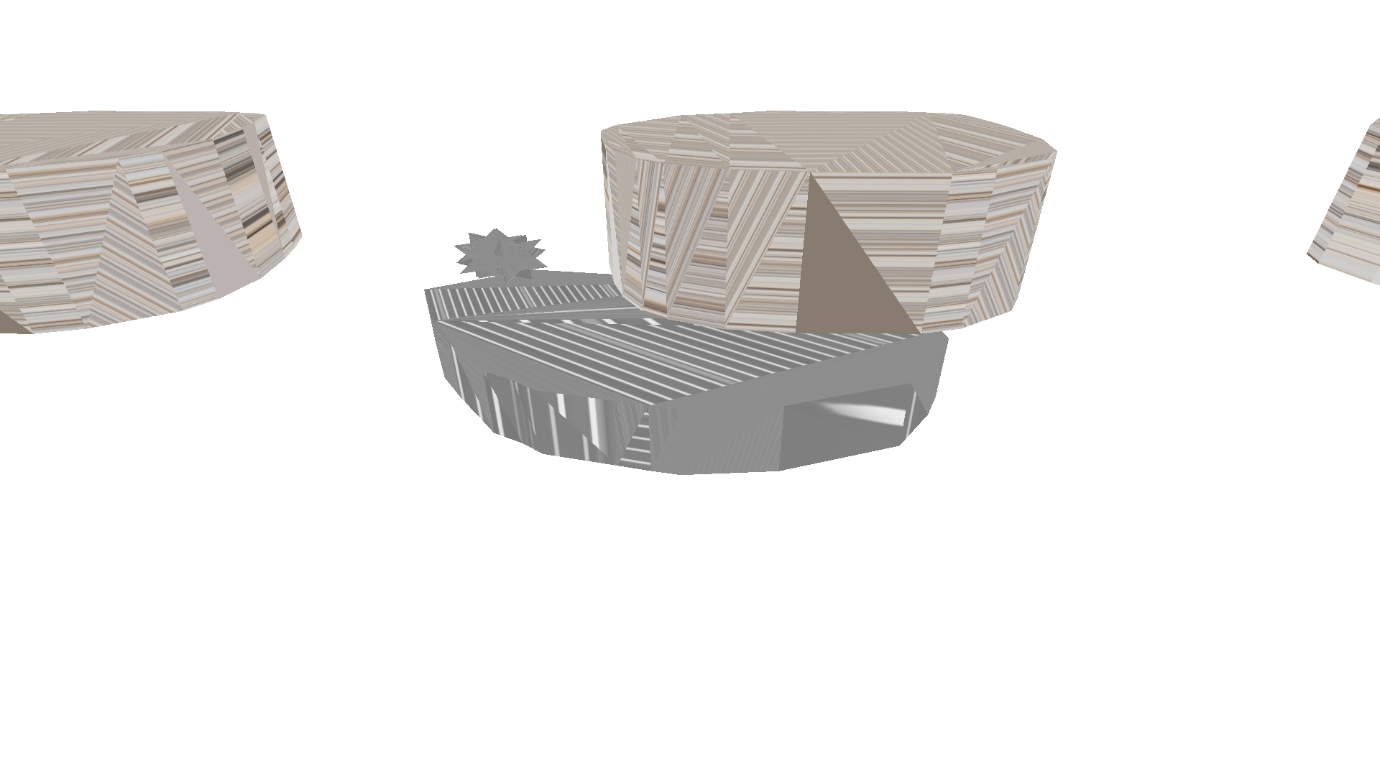
### Feature Test 1

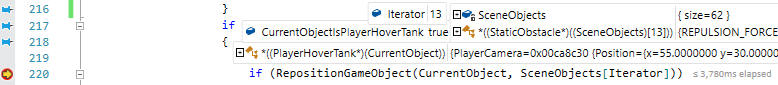
This test is simply for making sure that all the game-scene objects are shown to the Player as expected, this is the case, as the Player will see the following when the game is initialised:

Therefore, this test is successful.

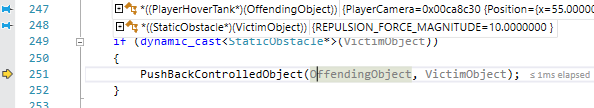
### Feature Test 2

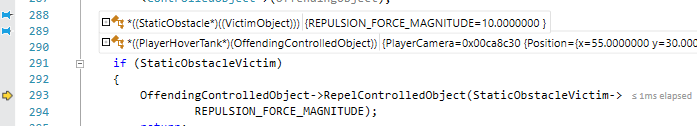
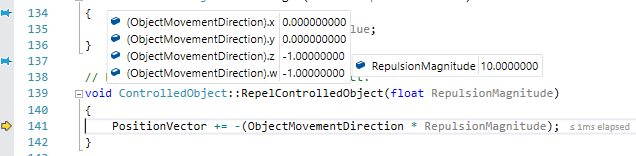
For this test, the Player’s hover-tank must be close enough to a Static-Rock, as shown here:



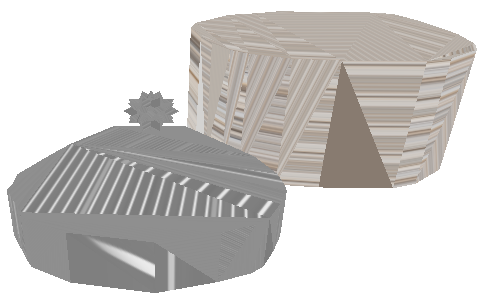
When they collide with the rock, the following breakpoint is triggered (within the GameScene’s ManageCollisionChecking() method):

From this, one can tell that the CurrentObject getting checked is the PlayerHoverTank, being checked against a GameObject at index 13 of the SceneObjects collection (that CurrentObject has collided with, after checking for bounding-sphere collision between it and this GameObject), which is a StaticRock. The program then traverses to the RepositionGameObject() method to handle repositioning CurrentObject appropriately:

In this function, the object receiving a collision (VictimObject) is a StaticObstacle whilst the object colliding with the VictimObject (OffendingObject) is a PlayerHoverTank, so in this situation, the PushBackControlledObject() method will be called:

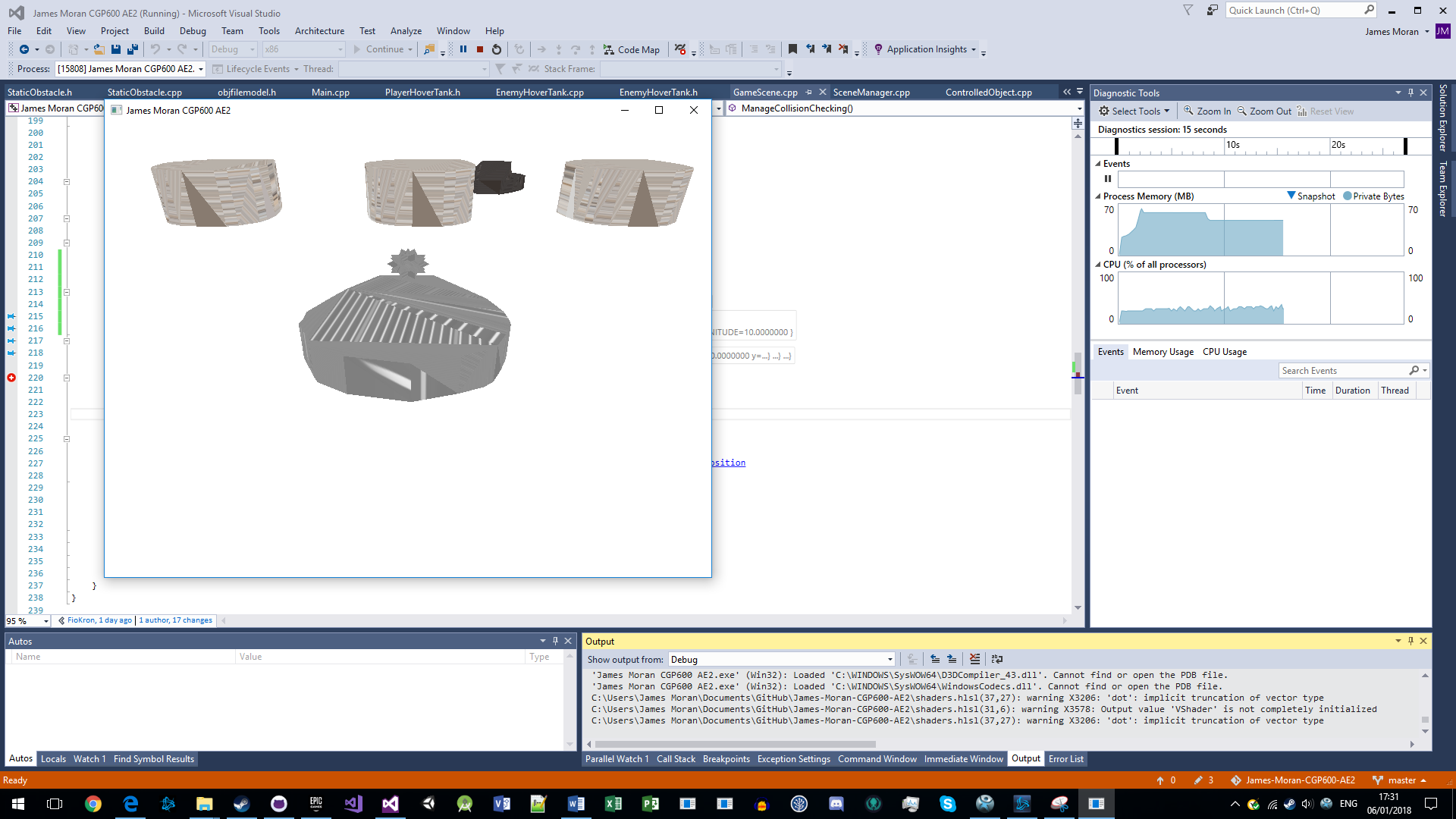
This method then makes sure which object-type is colliding with which other object type to perform the correct course of action, in this case, repel the Player’s hover-tank from this StaticObstacle via the RepelControlledObject() method:

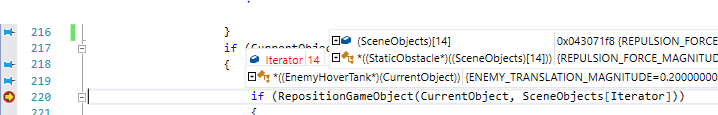
This method simply adds the (negated) movement direction of this object multiplied by the parsed-in RepulsionMagnitude, to cause the ControlledObject to bounce back appropriately, away from this StaticObstacle:

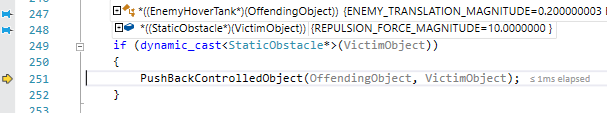
So therefore, this test is successful.

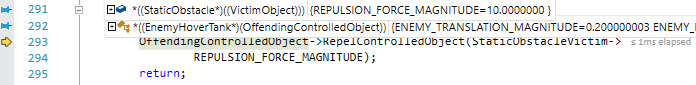
### Feature Test 3

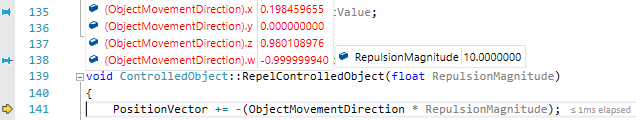
For this test, the Enemy’s hover-tank must be close enough to a StaticRock, as shown here:

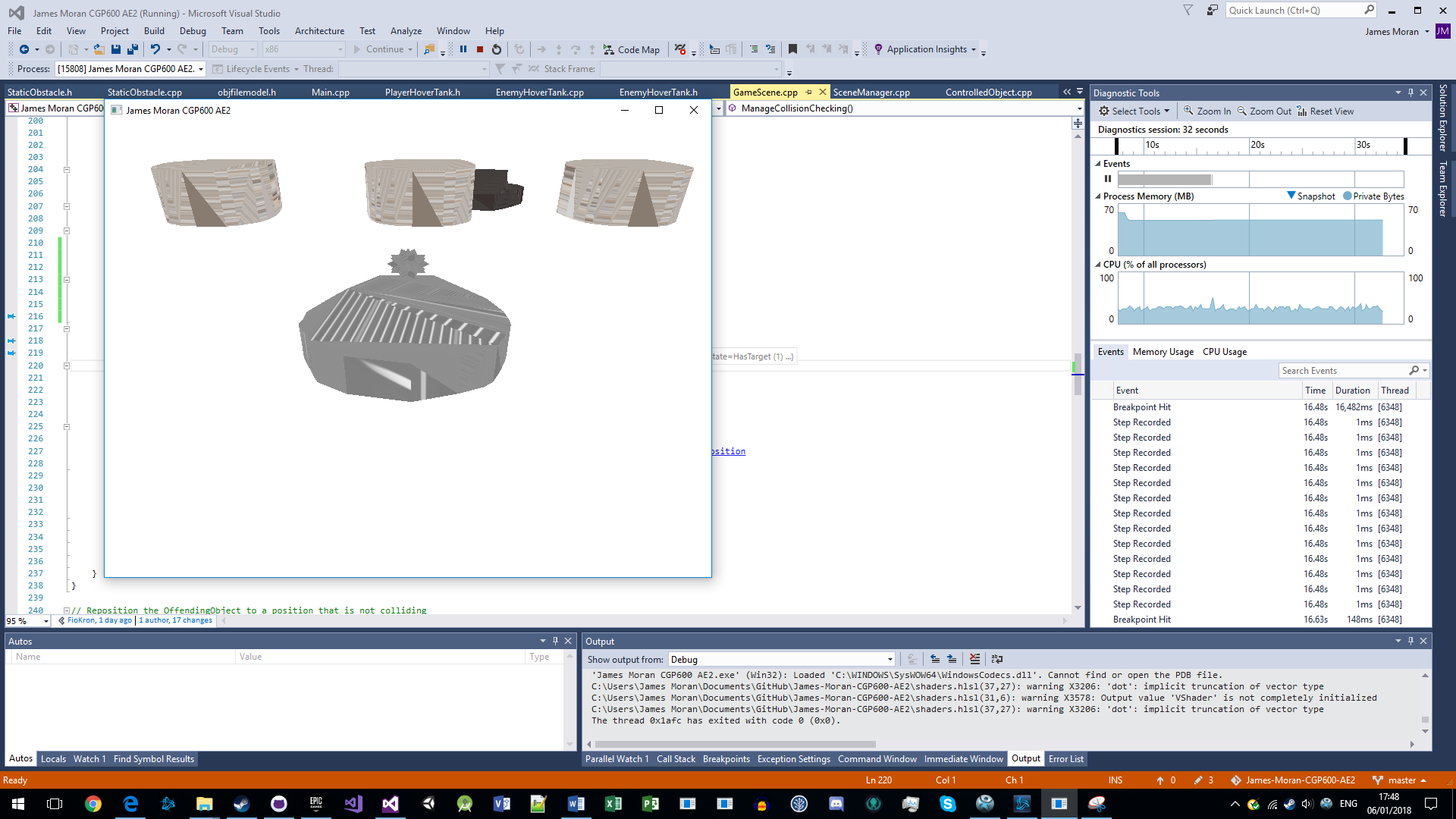
 When they collide with the rock, the breakpoint as shown on the next page is triggered:

So once again, RepositionGameObject() is called (with CurrentObject as an EnemyHoverTank this time):

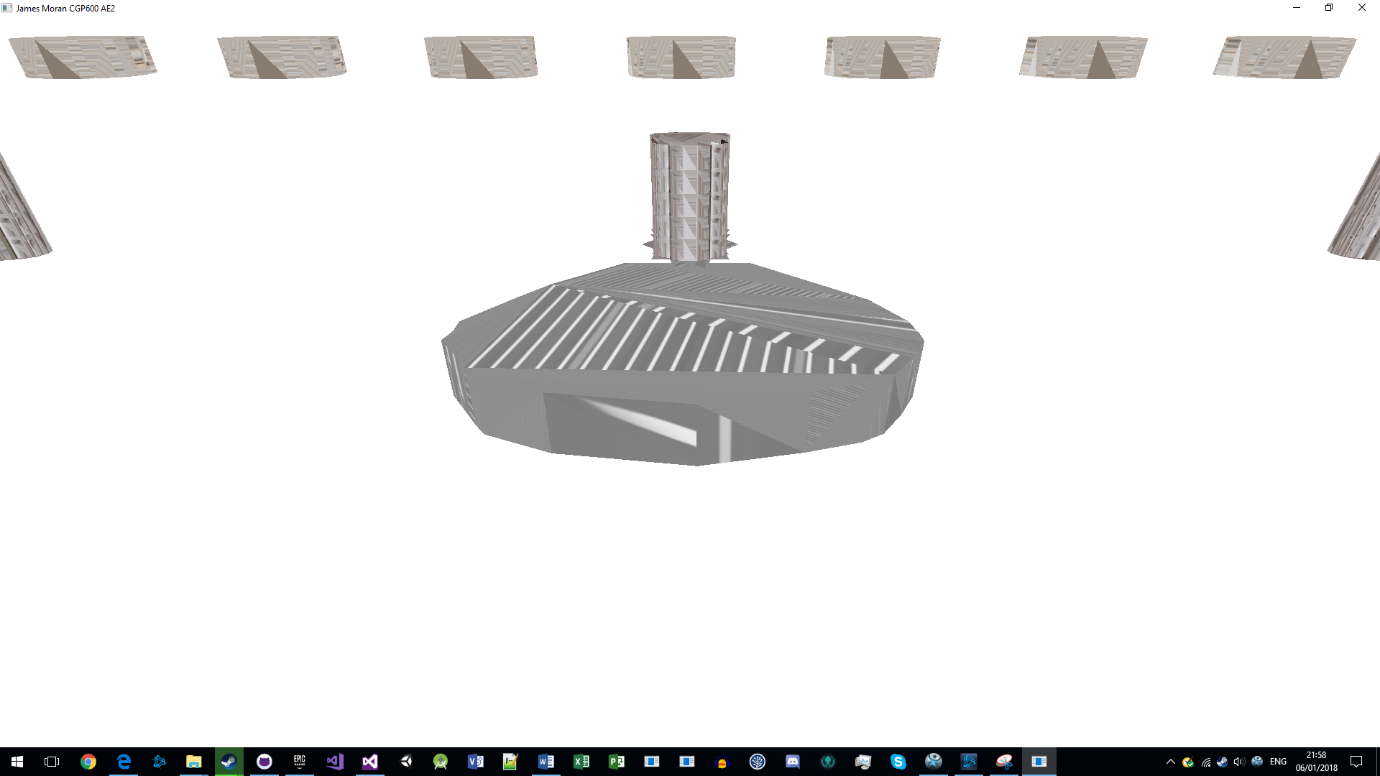
Which then calls the PushBackControlledObject() method:

Which then calls the RepelControlledObject() method (for the EnemyHoverTank):

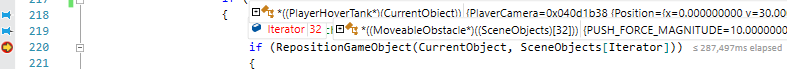
Which prevents the Enemy’s hover-tank from getting near to the Player through this StaticRock:

Therefore, this test is successful.

### Feature Test 4

For this test the Player’s hover-tank must be near a Wooden-Barrel:

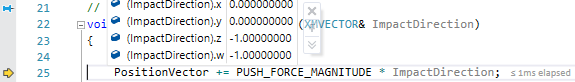
On the next page is the breakpoint triggered once again, in the GameScene’s ManageCollisionChecking() method:

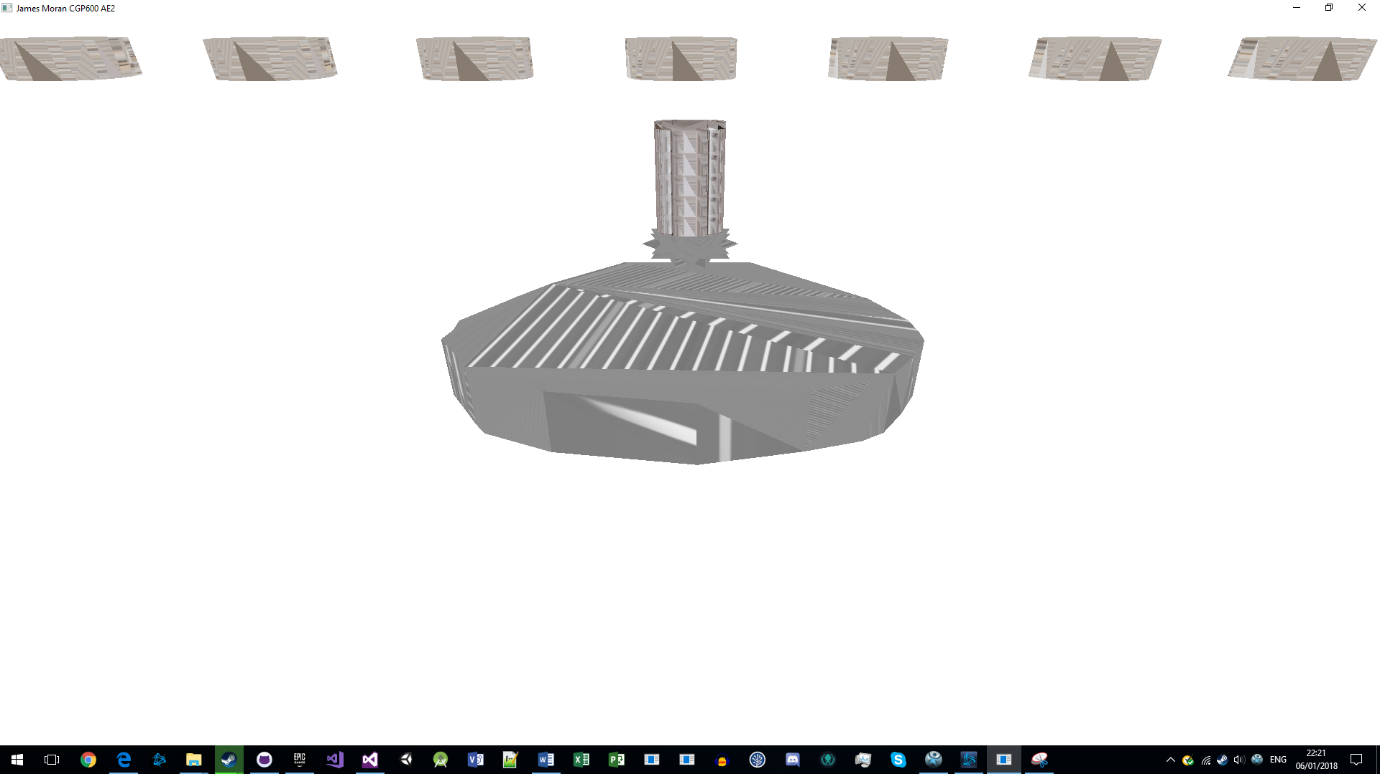
Which then calls RepositionObject():

Which then calls PushAwayMoveableObstacle (as the VictimObject in this case is a MoveableObstacle):



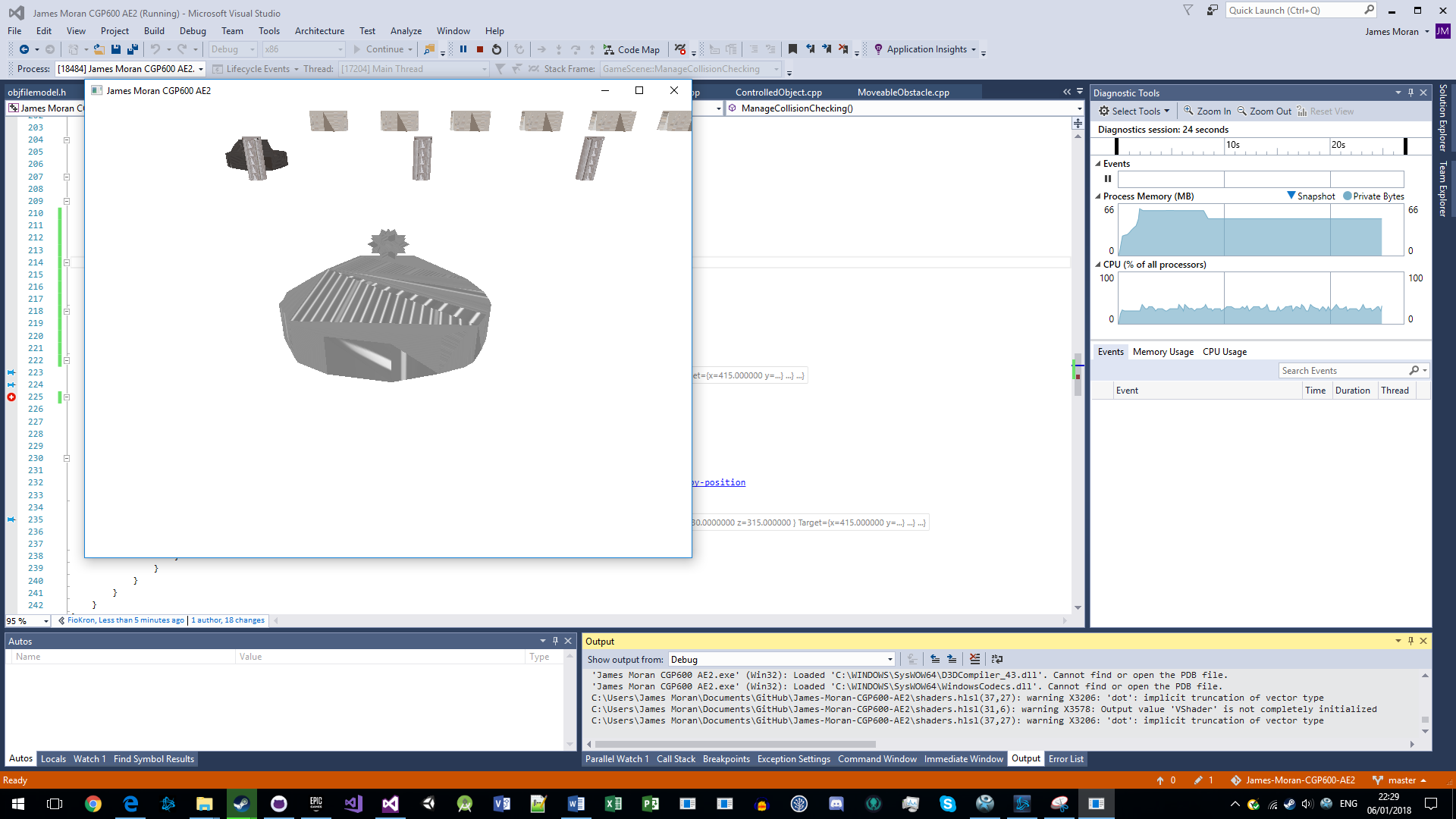
Which then calls the OnImpact() method of the MoveableObstacle (parsing in the ControlledObject’s MovementDirection):

Which results in the MoveableObstacle getting moved by the respective direction of impact:

This Wooden-Barrel is no longer in line with the other barrels, so this test is successful.

### Feature Test 5

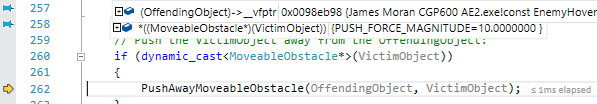
For this test, an EnemyHoverTank must be near to a MoveableObstacle:



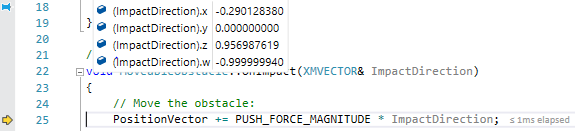
Then the same breakpoint (with different conditions) is triggered when they collide with this WoodenBarrel (as shown on the next page):

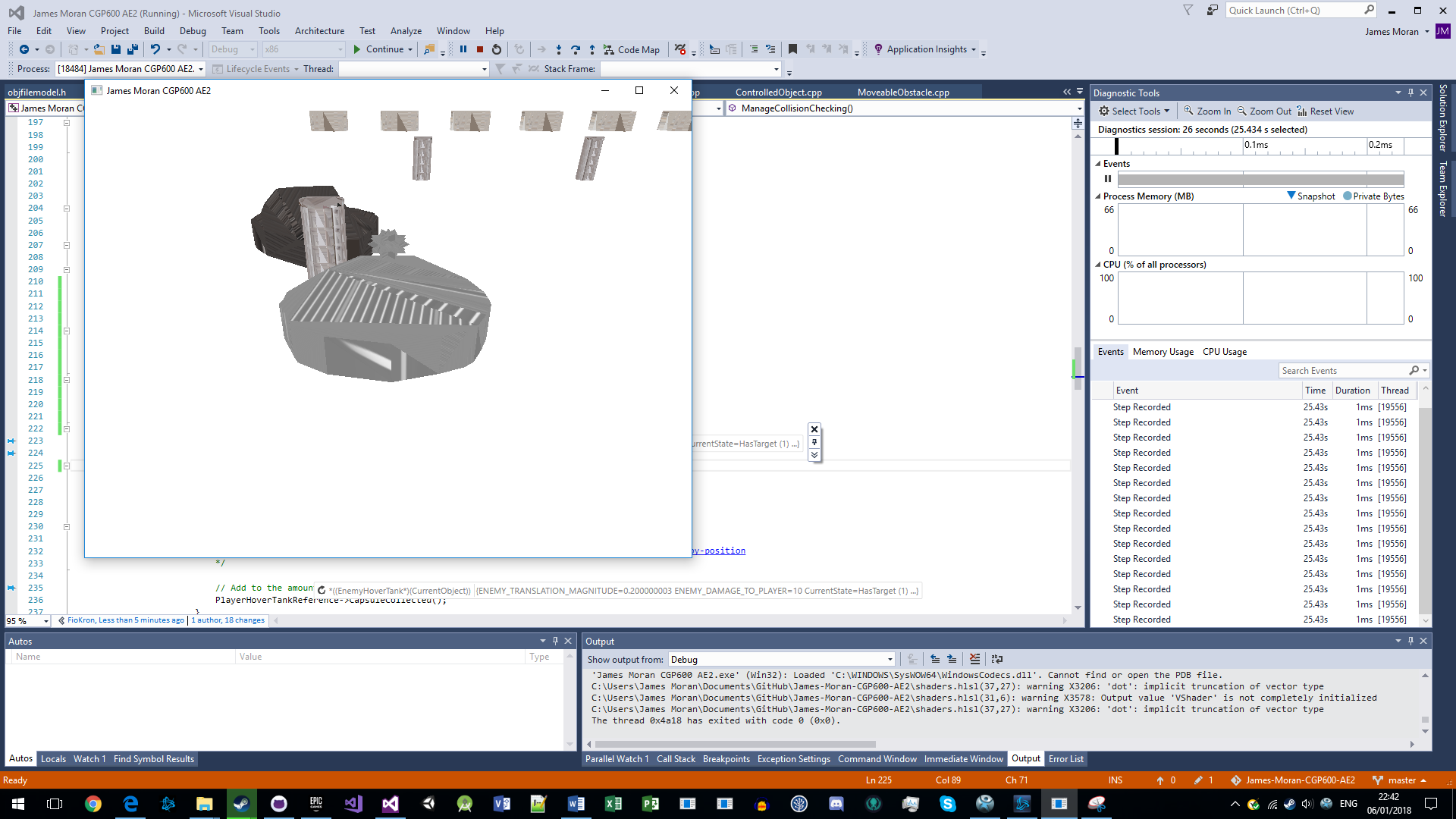


Which then calls RepositionObject():

Which then calls PushAwayMoveableObstacle():

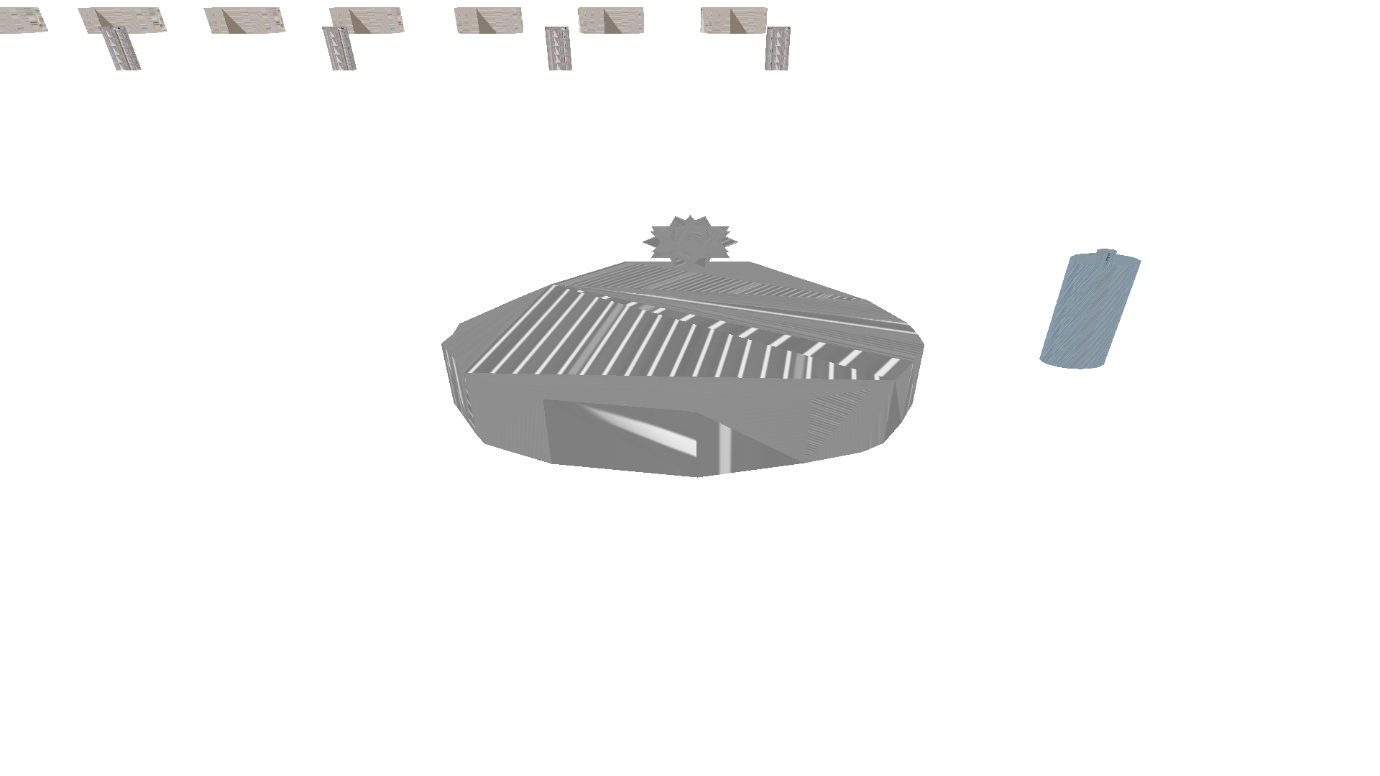
Which then calls OnImpact():

Which then results in movement of the Wooden-Barrel:

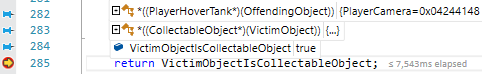
Therefore, this test is successful.

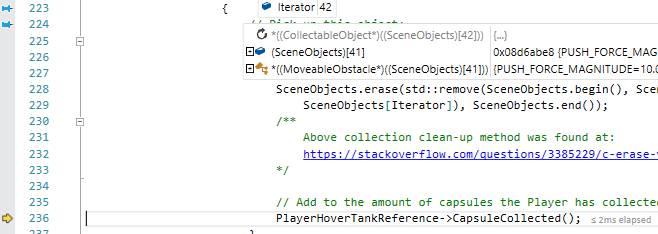
### Feature Test 6

To initialise this test, the Player is to have collected all but 1 of the Energy-Capsules in the level and the Player’s hover-tank is near to the last Energy-Capsules of the level:

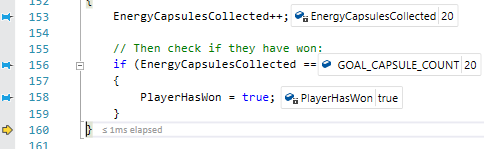


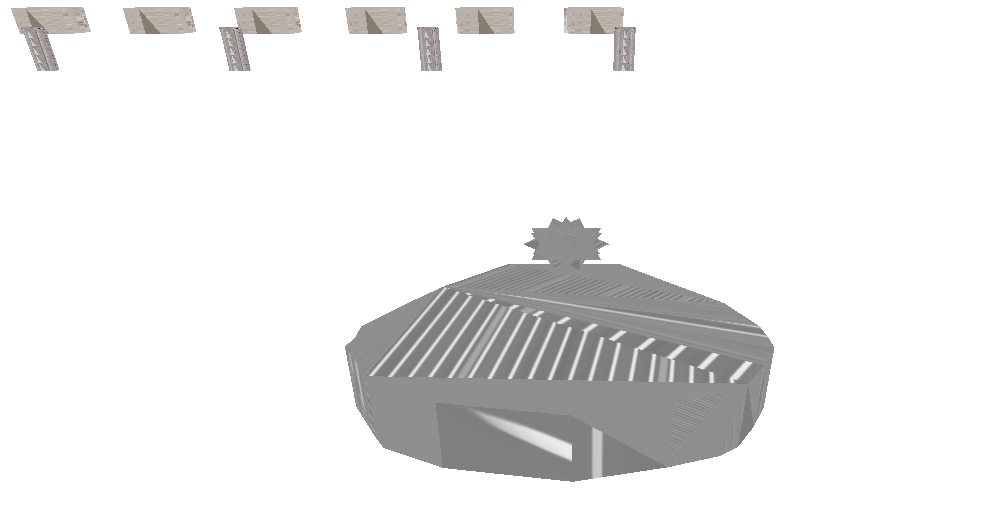
When the Player collides with the last Energy-Capsule, the following breakpoint is triggered (within RepositionGameObject):



This value is then checked in ManageCollisionChecking() and given it is true, the Energy-Capsule is removed from the collection using the methods in this screenshot:

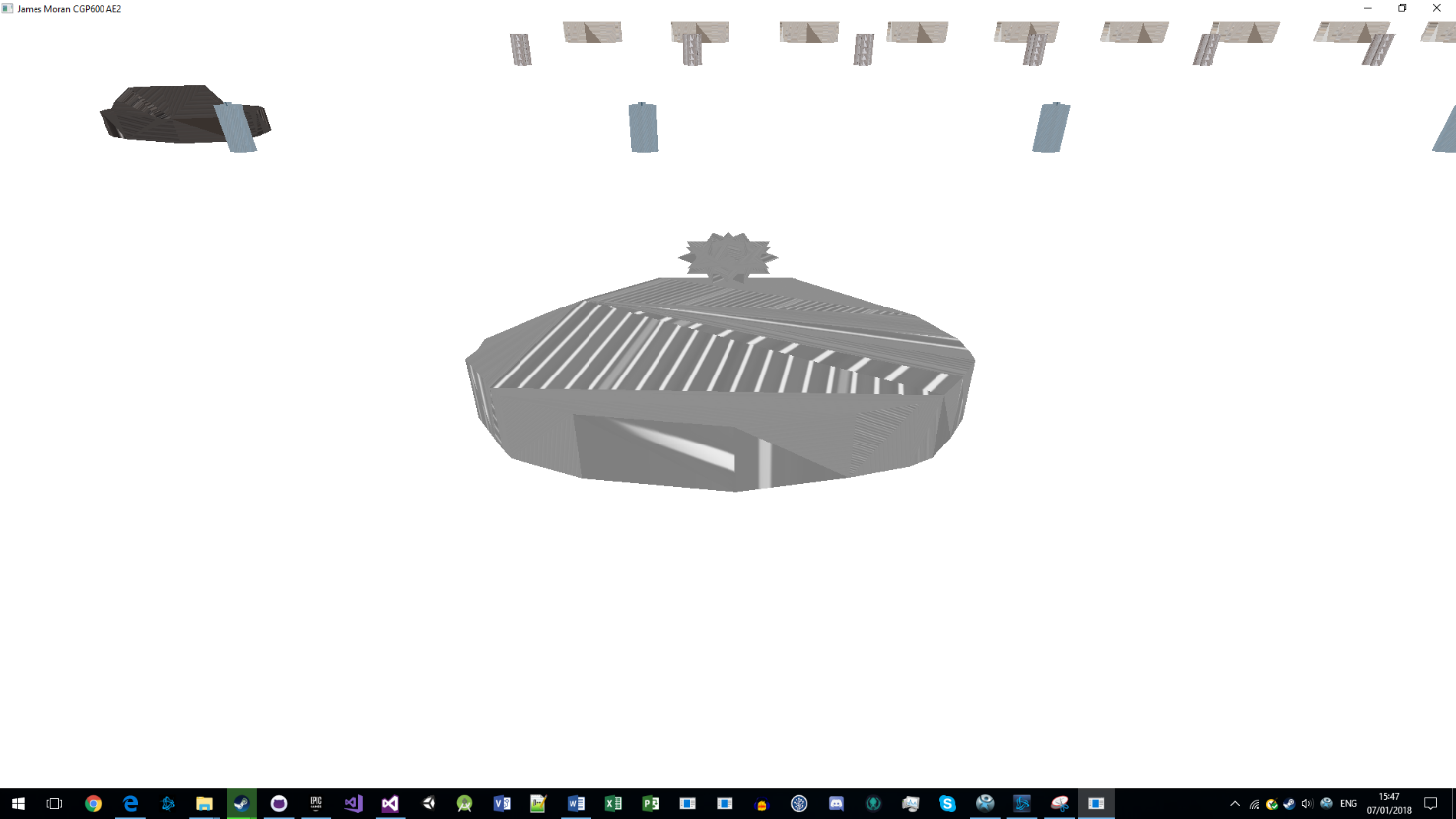
(The Energy-Capsule was at position 42 in SceneObjects, but as it has now been removed, that index of the vector no longer has a value (as the vector has a reduced size) and the last value in SceneObjects is a MoveableObstacle). Then CapsuleCollected() is called:

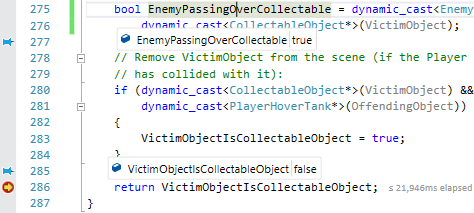
Which then causes the Player to win the game, with a freeze-frame of their hover-tank shown to the Player for 3 seconds, before the applications termination (with the last Energy-Capsule no longer present in the game-scene):

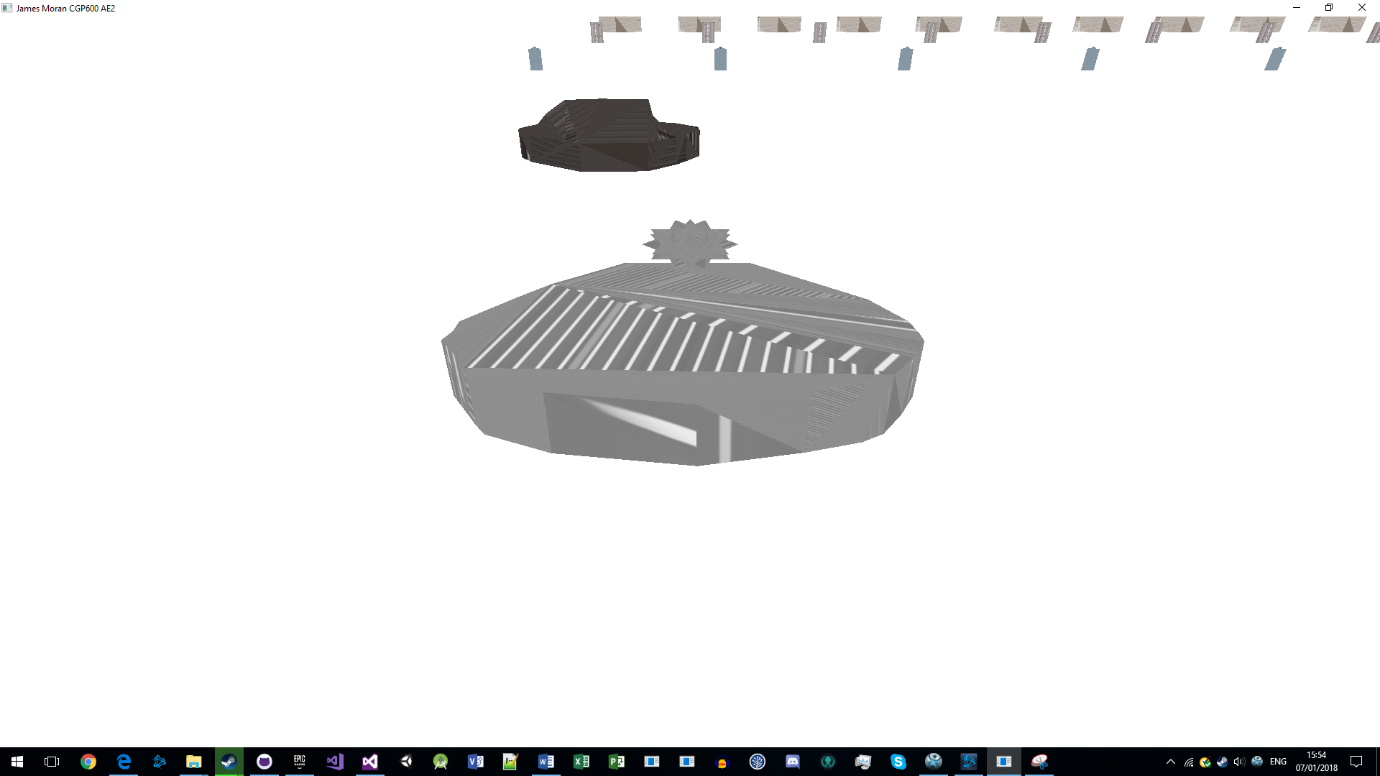
Therefore, this test is successful.

### Feature Test 7

For this test, the Enemy’s hover-tank must be near an Energy-Capsule:

Then, the same breakpoint (with different conditions), for the previous test is triggered:

 As they are an EnemyHoverTank, the Energy-Capsule will not be collected, resulting in them passing through it, instead of removing it from the collection (screenshot showing this is on the next page):

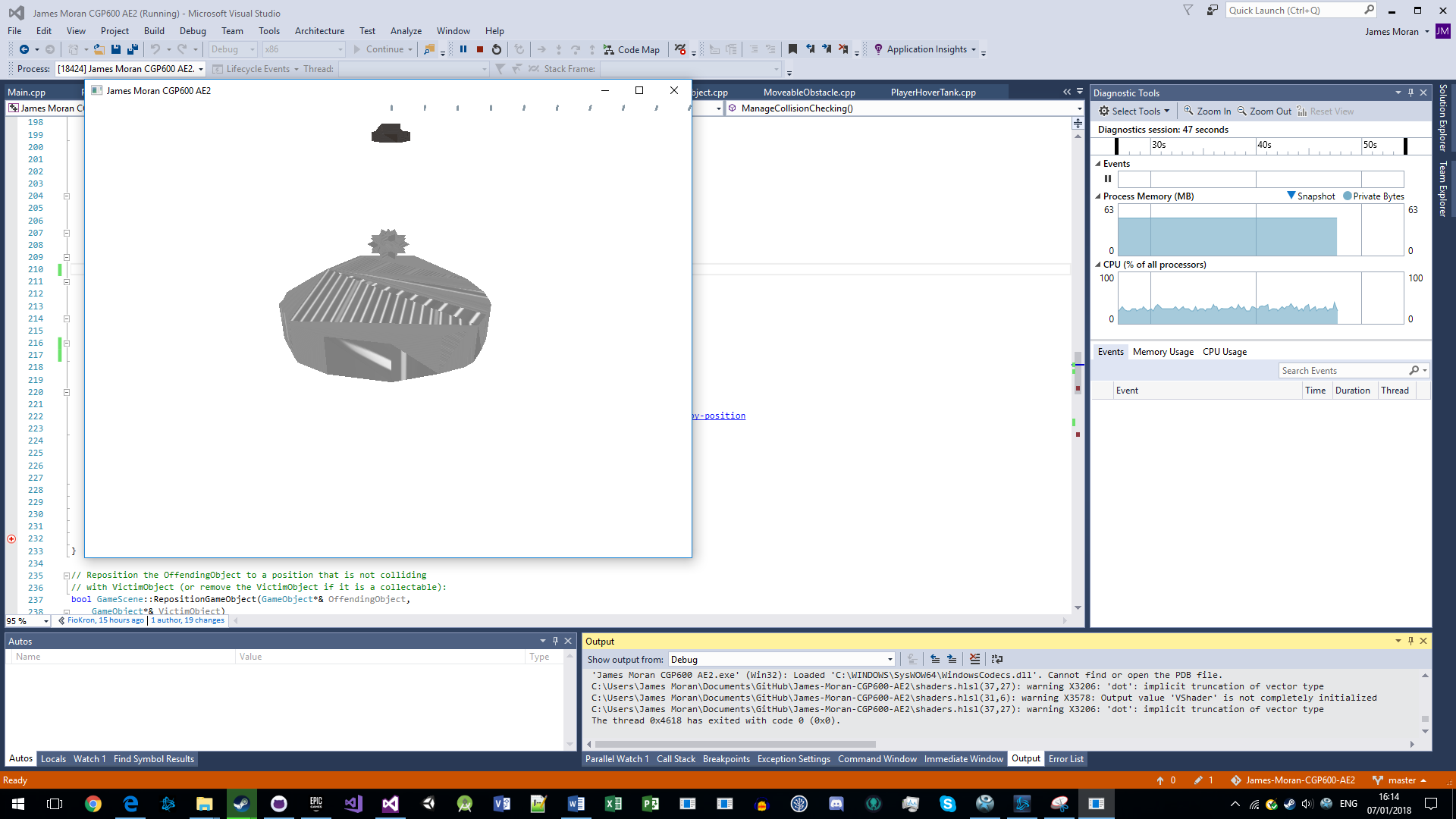
Therefore, this test is successful.

### Feature Test 8

As can be seen from the previous tests, the Enemy’s hover-tank will rotate to the face the Player’s hover-tank, then move towards them in a straight line, so this test is also successful.

### Feature Test 9

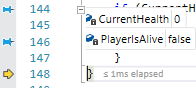
For this test, the Enemy’s hover-tank must be lined up with the Player’s hover-tank, soon to collide with it:



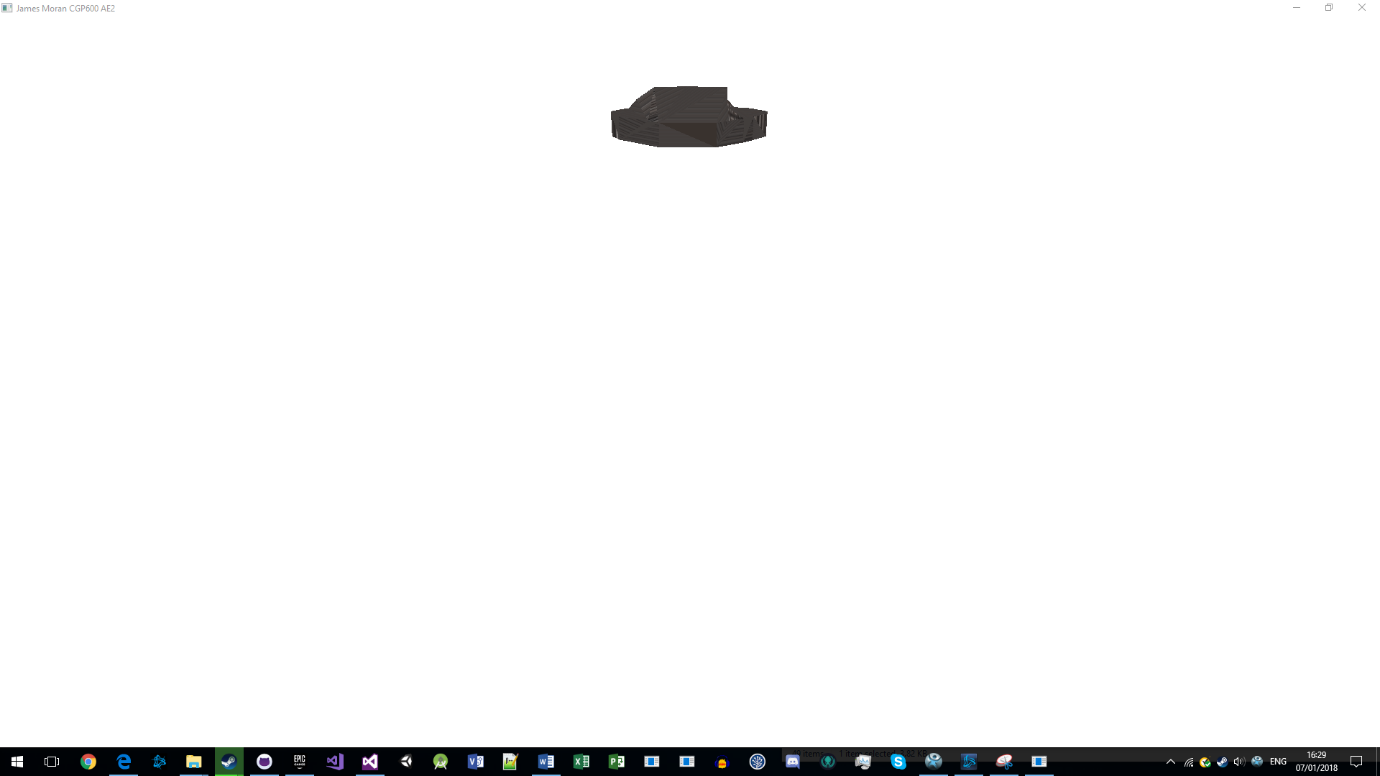
After triggering a breakpoint in HandleCollisionChecking(), program-flow transitions into the PlayerHoverTank’s ModifyHealth() method:

OAs the Player was at 10 health (before taking 10 health-points of damage), this test is successful and flows into the final test.

### Feature Test 10

Given where program execution was stopped after the previous test’s conclusion, as the Player is at 0 health-points, they are no longer alive:

So, on the next page, one can see that the Player’s hover-tank is no longer displayed:

The game will then freeze for 3 seconds, before terminating itself, as expected, so this test is successful, ending the last of the Feature Tests.

Basic Testing is in the next section of this appendix:

## Basic Testing

Testing for each of the basic requirements, is as documented in this table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Test Description | Expected Result | Actual Result | Success or Failure |
| 1 | The Player should be able to move around the environment. | The Player can move around the environment. | The Player can move around the environment. | Success. |
| 2 | The environment should be quite large and must have static and moving obstacles. These must be models, with some of these objects able to be pushed by the Player and some of these objects being collectable. | The environment has static, moveable and collectable objects, whilst being quite large (with the Player’s view distance at 1000 units) and no bounds on where the Player can move to in this environment. | The environment has static, moveable and collectable objects, whilst being quite large (with the Player’s view distance at 1000 units) and no bounds on where the Player can move to in this environment. | Success. |
| 3 | The objects in the environment must have textures and some form of lighting. | The objects have textures and lighting showing on them. | The objects have textures but no discernible lighting present. | Failure: No lighting on scene-objects |
| 4 | The Player should collide with objects, either stopping for static-objects, moving the object if it is moveable and picking up collectable objects. | The Player collides with all object types, getting knocked back slightly for static obstacles, moving the object accordingly for moveable objects and picking-up collectable objects. | The Player collides with all object types, getting knocked back slightly for static obstacles, moving the object accordingly for moveable objects and picking-up collectable objects. | Success. |
| 5 | The environment must contain non-player entities represented by a model, or at least a series of connected 3D shapes, using textures and lighting. | The Enemy’s hover-tank is visible in the environment (with a model, texture and lighting), after they get within line of sight to them/they get within the Player’s line of sight. | The Enemy’s hover-tank is visible in the environment (with a model and texture, but no discernible lighting), after they get within line of sight to them/they get within the Player’s line of sight. | Failure: No discernible lighting on Enemy-Hover-Tanks. |
| 6 | The entities should move around the environment in some fashion. | The Enemy’s hover-tank will rotate to face the Player and move towards them in a straight line, continuously. | The Enemy’s hover-tank will rotate to face the Player and move towards them in a straight line, continuously. | Success. |
| 7 | The entities should collide with objects. This should cause the entities to perform an action. | The Enemy’s hover-tank will stop for static-obstacles, push moveable-obstacles along with them and simply pass through Energy-Capsules. | The Enemy’s hover-tank will stop for static-obstacles, push moveable-obstacles along with them and simply pass through Energy-Capsules. | Success. |
| 8 | There should be some form of interaction between the non-player entities and the Player based on collision detection. | The Enemy’s hover-tank damages the Player’s hover-tank, when it collides with them. | The Enemy’s hover-tank damages the Player’s hover-tank, when it collides with them. | Success. |

## Evidence for Basic Testing

### Basic Test 1

For this test, as the Player is required to move around the environment for Feature Tests 2, 4 and 6, this test is also successful.

### Basic Test 2

For this test, as per the screenshot of Feature Test 1, one can see the Player’s hover-tank, StaticObstacles, MoveableObstacles and CollectableObjects, that are in the Player’s line of sight when they start the level, so this test is also successful.

### Basic Test 3

For this test, although objects are visible in the environment with an appropriate texture, there is no discernible lighting emanating from them, so this test has failed.

### Basic Test 4

For this test, as per Feature Tests 2, 4 and 6 (for checking the Player hover-tank’s interaction with StaticObstacles, MoveableObstacles and CollectableObjects respectively), as those tests has succeeded, this test has also succeeded.

### Basic Test 5

For this test, as per Feature Tests 3, 5, 7, 8, 9 and 10, there is a model to represent Enemy hover-tanks, with an appropriate texture, but no discernible lighting. So unfortunately, this test has failed on that basis.

### Basic Test 6

For this test, as per Feature Test 8 (referencing other feature tests), the Enemy’s hover-tank will rotate to face the Player, then move towards them in a straight line, continuously. Thus, this test is successful.

### Basic Test 7

For this test, as per Feature Tests 3, 5 and 7, the Enemy’s hover-tank will act accordingly for each type of scene-object (getting knocked back from static-obstacles, pushing moveable-obstacles as per their movement and not collecting Energy-Capsules when they pass over them). This test has therefore, succeeded.

### Basic Test 8

For this test, as per Feature Test 9 (leading into Feature Test 10), the Player will take damage when an Enemy’s hover-tank collides with them, leading to them losing the game if this is the 10th time they have been struck (as per Feature Test 10). Therefore, this final Basic Test has succeeded.

# References

1. © cplusplus.com, 2000-2017. *assert* [viewed on the 29/11/2017]. Available from: <http://www.cplusplus.com/reference/cassert/assert/>
2. Matthew Hoggan, 2012. *Reason to Pass a Pointer by Reference in C++?* [viewed on the 30/11/2017]. Available from: <https://stackoverflow.com/questions/10240161/reason-to-pass-a-pointer-by-reference-in-c>
3. gradbot, 2012, *Macro definition clash between directx headers and winerror.h* [viewed on the 07/12/2017]. Available from: https://stackoverflow.com/questions/12663091/macro-definition-clash-between-directx-headers-and-winerror-h
4. © Microsoft 2018. *DXGI Overview* [Viewed on the 08/01/2018]. Available from: <https://msdn.microsoft.com/en-us/library/windows/desktop/bb205075%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396#Handling_Window_Resizing>
5. © Microsoft 2018. *IDXGISwapChain::ResizeBuffers method* [Viewed on the 08/01/2018]. Available from: <https://msdn.microsoft.com/en-us/library/bb174577(v=vs.85).aspx>
6. Master Kenneth, 2014, DirectX Leak Debugging [viewed on the 07/12/2017]. Available from: <http://masterkenth.com/directx-leak-debugging/>
7. Mike Celestino, 2015. *Lego Millennium Falcon Top-Down View* [digital image] [viewed on the 09/12/2017]. Available from: http://www.insidethemagic.net/2015/12/enormous-custom-lego-millennium-falcon-proves-star-wars-fans-dedication-attention-to-detail/
8. iedoc, 2015. 32. Simple 3rd Person Camera [Viewed on the 24/12/2017]. Available from: <https://www.braynzarsoft.net/viewtutorial/q16390-32-simple-3rd-person-camera>
9. Allen Sherrod and Wendy Jones, 2012. *Beginning DirectX® 11 Game Programming.* Boston, MA: Course Technology PTR
10. John McGrath and James Moran, 2017. *CGP600 Advanced Games Programming AE1 – Group Project.* Southampton: Southampton Solent University.
11. © 1970-2017 Analytical Scientific, LTD, 2017. *Pumice Rock* [Digital Image] [Viewed on the 26/12/2017]. Available from: <https://www.analyticalsci.com/store/p432/Pumice_Rock.html>
12. Copyright © 2005-2017, Textures.com. *RockSmoothErosion0027* [Digital Image] [Viewed on the 28/12/2017]. Available from: <https://www.textures.com/download/rocksmootherosion0027/68006?q=rock&filter=seamless> (TexturesCom\_RockSmoothErosion0027\_1\_seamless\_S.jpg)
13. Copyright © 2005-2017, Textures.com. *Bare Metal Threadplate Floor (S0070)* [Digital Image] [Viewed on the 29/12/2017]. Available from: <https://www.textures.com/preview/substance0070/126684?q=plate>

(TexturesCom\_BareThreadplateFloor\_1024\_height.TIFF)

1. Independent.co.uk, 2014. *Jack Daniels Whiskey Barrel* [Digital Image] [Viewed on the 30/12/2017]. Available from: <https://static.independent.co.uk/s3fs-public/thumbnails/image/2014/08/28/12/whiskey%20barrel.jpg>
2. Copyright © 2005-2017, Textures.com. *WoodStudded0044* [Digital Image] [Viewed on the 30/12/2017]. Available from: <https://www.textures.com/download/woodstudded0044/46382?q=wood&filter=seamless> (TexturesCom\_WoodStudded0044\_1\_seamless\_S.jpg)
3. 9to5Google.com, 2013. *Huge Improvements to batter-life expected in Android 4.3?* [Digital Image] [Viewed on the 31/12/2017]. Available from: <https://9to5google.files.wordpress.com/2013/07/glowing-green-battery-charging.jpg>
4. Copyright © 2005-2017, Textures.com. MetalPlatesPainted0022 [Digital Image] [Viewed on the 01/01/2018]. Available from: <https://www.textures.com/download/metalplatespainted0022/45883?q=can> (Textures.com.TexturesCom\_MetalPlatesPainted0022\_1\_seamless\_S.jpg)
5. Georg Fritzsche, 2010. *C++ Erase vector element by value rather than by position? [duplicate]* [viewed on the 01/01/2018]. Available from: <https://stackoverflow.com/questions/3385229/c-erase-vector-element-by-value-rather-than-by-position>
6. Anthony Carmona, Copyright © 2005-2018, Textures.com. Metal Threadplate Floor (S0035) [Digital Image] [Viewed on the 02/01/2018]. Available from: <https://www.textures.com/download/substance0035/127550?q=black+metal>

(TexturesCom\_ThreadplateFloor\_1024\_albedo.tiff)