James Moran

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

James Moran CGP600 AE2 – Individual Project Report

Contents

# 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

4 14 15 16 17

* The Player is to be represented by a static-mesh (as shown in Fig. 4 of Appendix A), that they 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 project the most advanced working version of the project (no discernible memory leaks), which is also stable. I went through the code, refactoring suitable functions/methods, as well as properties, into classes, with respects to which actions these methods/functions perform and which class would require the respective properties for such.

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

First off, came the process of resolving macro-definition classes between winerror.h and dxgi.h. Although a process is listed on MSDN, to resolve this issue, I chose the answer that was not the accepted answer (with the second highest amount 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 similar to those listed in Appendix A: Fig. 1.

I resolved live objects not being cleaned-up (which is what these messages indicate), by following all of 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, in order 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 ship, 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 ship. (Mike Celestino, 2015)

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

The second requirement is: ‘The player should be able to move around an environment’. For this purpose, I would want the camera behind the Player, looking down at them for a 3rd person perspective, whilst following them. I would also have wanted to place a (simple) static obstacle in the scene, that can be used to verify that the Player is moving.

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. The third requirement is: ‘The environment should be quite large, and must have static and moving obstacles (e.g., blocks, statues, and rolling rocks). These must be models (though they can be simple if you like). Some of these objects should be able to be pushed by the player, and others should be collectable.’ 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 ship, 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. The fourth requirement is: ‘The objects in the environment should have textures and some form of lighting.’. 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 this requirement: ‘The player should collide with objects; either stopping for static objects, or pushing them if they are moveable, or pick them up if they are collectable.’. 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 distinquish 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 fairly 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 of 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 particular scene, but only when it is active.

# 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 ship:



(Mike Celestino, 2015)

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

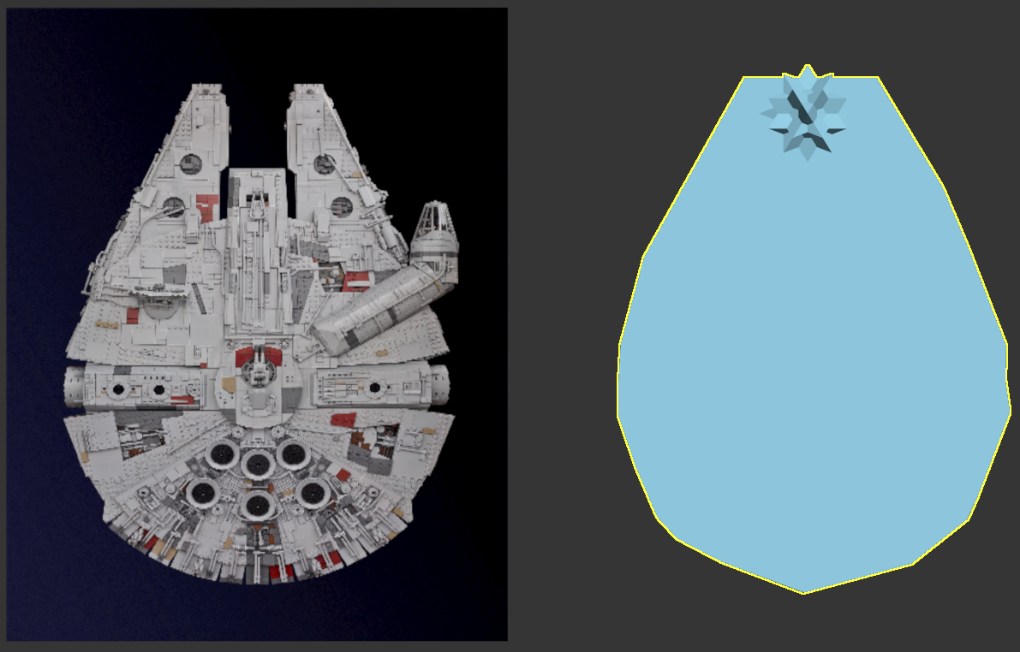


Figure 4: A perspective-view of the ship’s static-mesh in the default game scene:

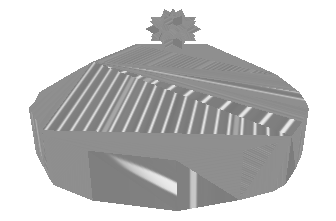


Figure 5: Side-on View of an igneous rock, used a reference image to produce static-obstacles in the default game scene:

(© 1970-2017 Analytical Scientific, LTD.)

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

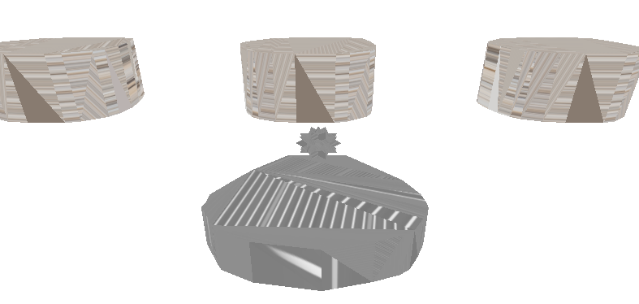


Figure 7: The image used to form a texture used for static-rocks in the default game-scene:

(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 (only the default scene at the moment):



(Copyright © 2005-2017, Textures.com)

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



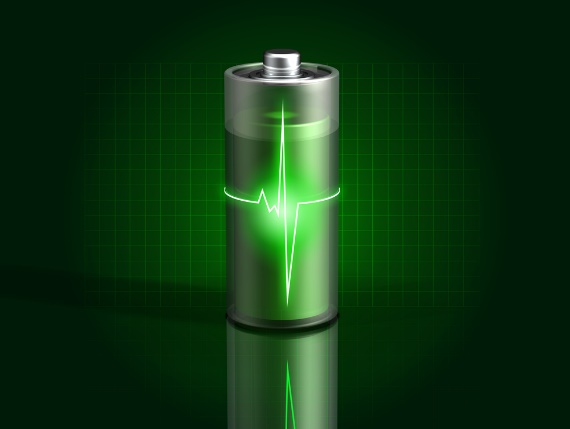
(Independent.co.uk, 2014)

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



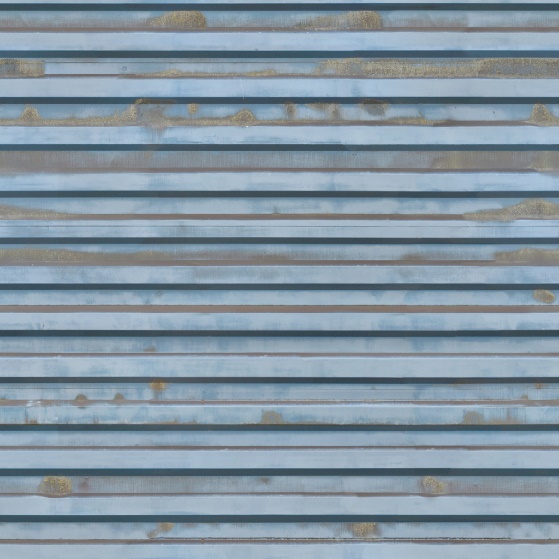
(Copyright © 2005-2017, Textures.com)

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

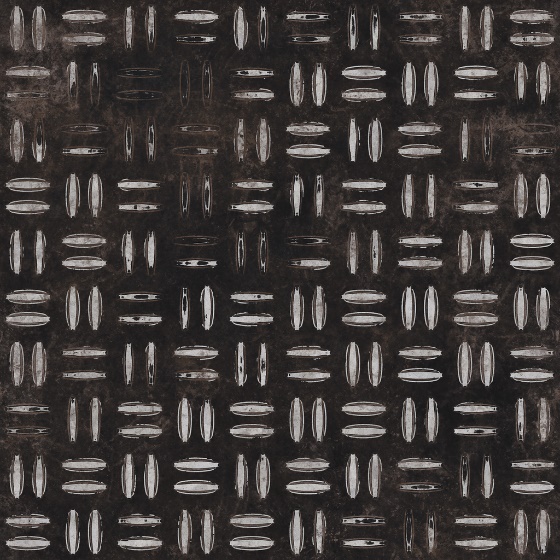


(9to5Google.com, 2013)

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:

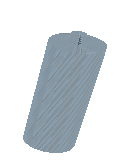
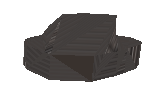


Figure 17: Enemy Hover-Tank Static-Mesh: To represent Enemy hover-tanks in the default game-scene:



# 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

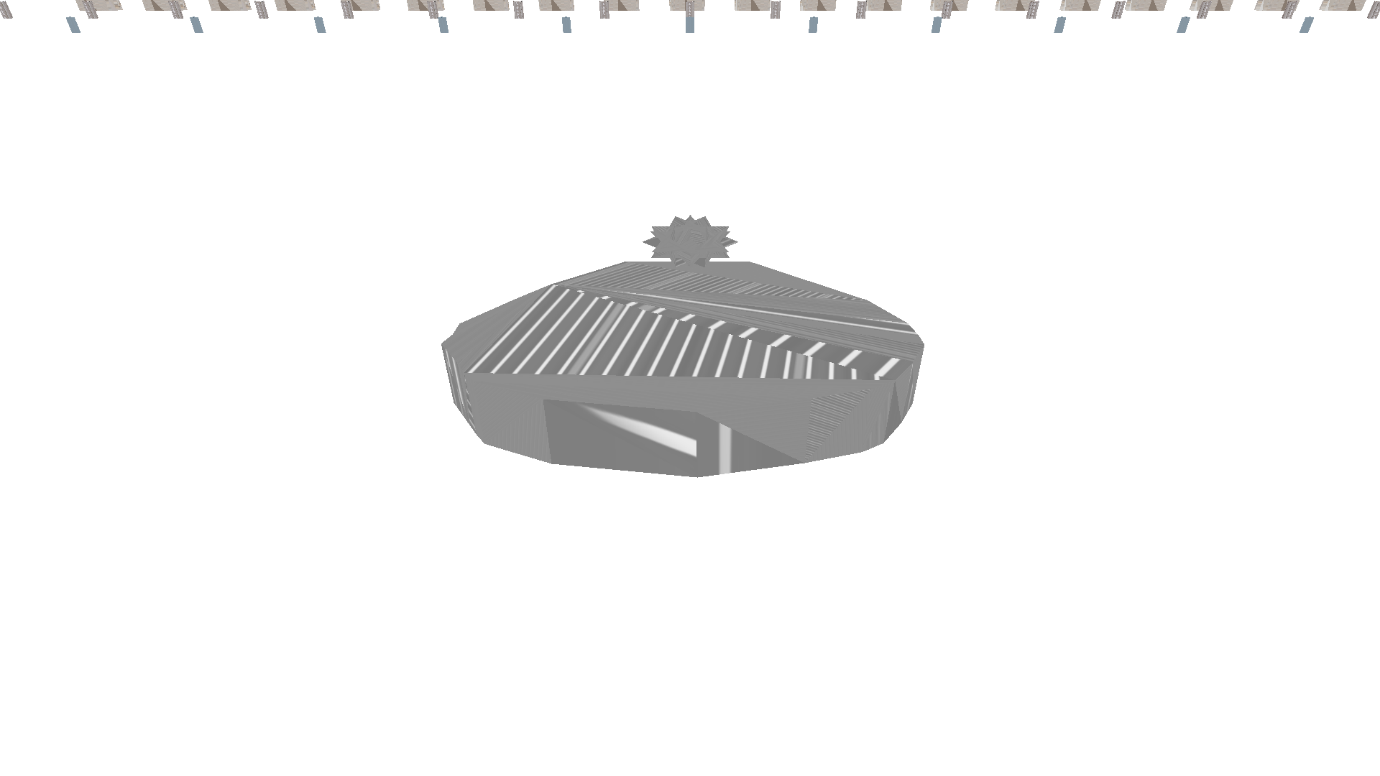
The tests of the features of Seek-and-Collect are documented here, as per the features outlined in the ‘Seek-and-Collect Features’ section of the introduction:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 a particular Enemy hover-tank’s current direction of movement. | Wooden-Barrels move along a particular 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

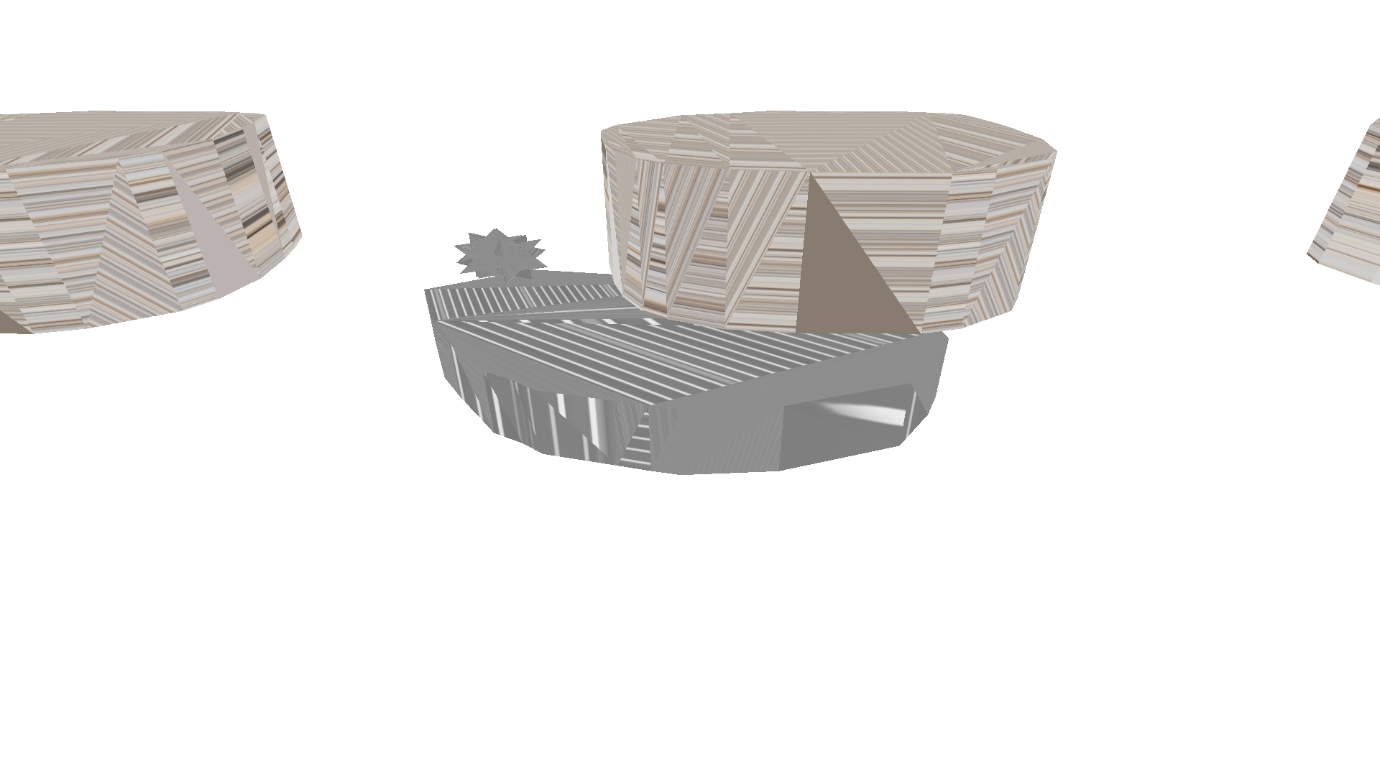
### Test 1

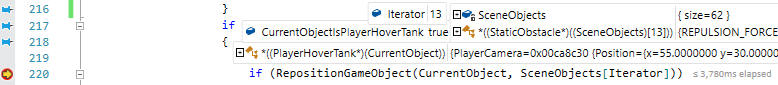
This test is simply for making sure that all the game-scene objects are shown to the Player as expected, this is the case, as the Player will see the following when the game is initialised:

Therefore, this test is successful.

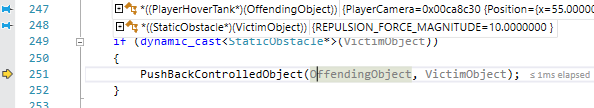
### Test 2

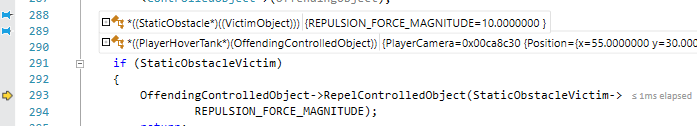
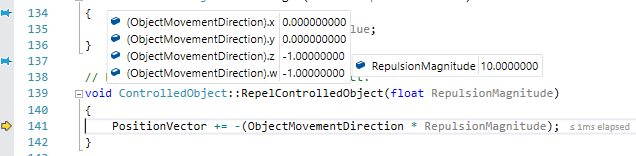
For this test, the Player’s hover-tank must be close enough to a Static-Rock, as shown here:



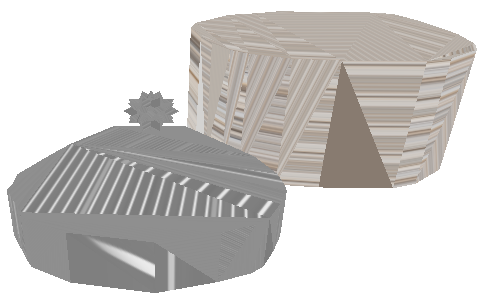
When they collide with the rock, the following breakpoint is triggered (within the GameScene’s ManageCollisionChecking() method):

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:

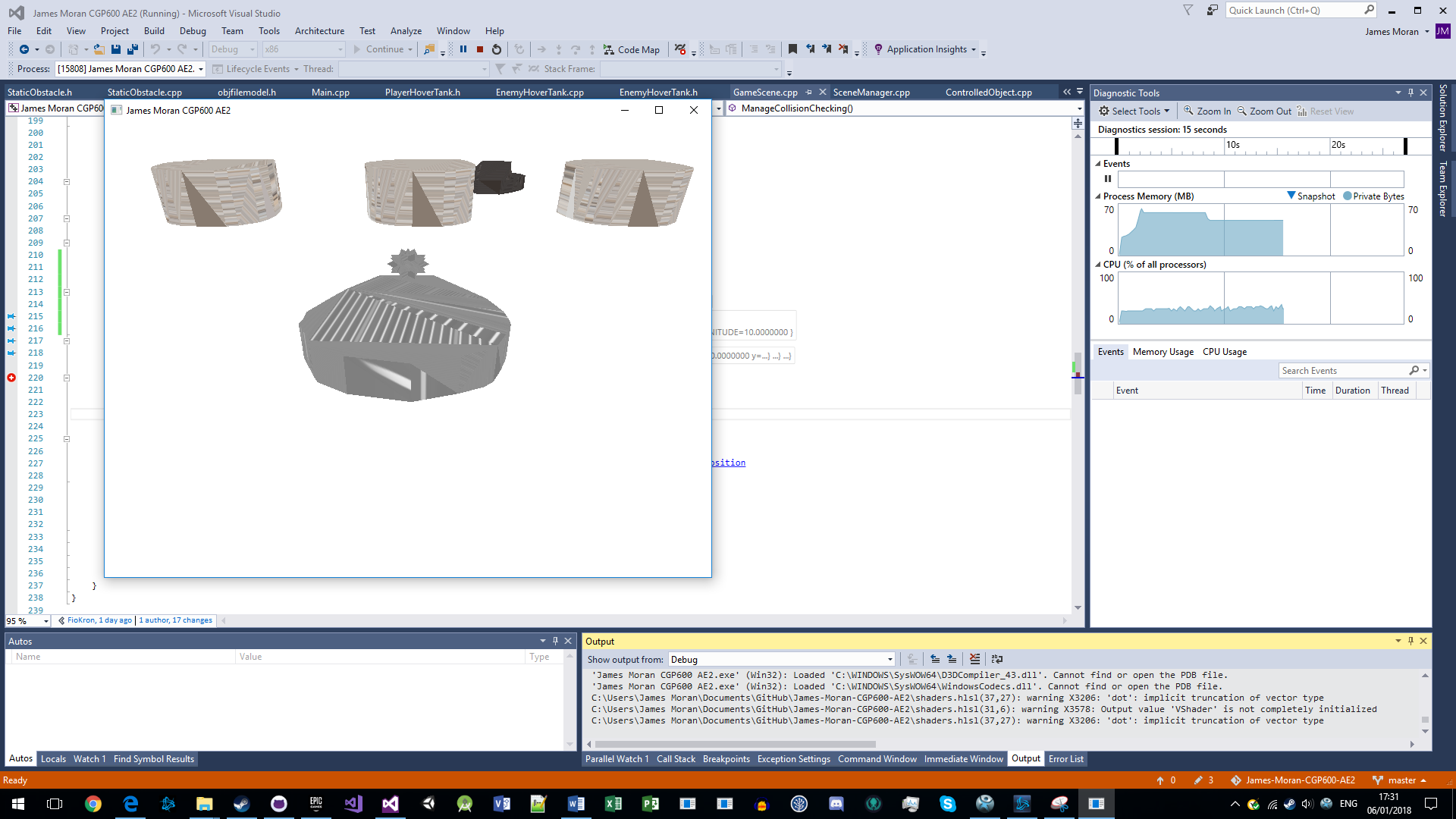
This method then makes sure which object-type is colliding with which other object type to perform the correct course of action, in this case, repel the Player’s hover-tank from this StaticObstacle via the RepelControlledObject() method:

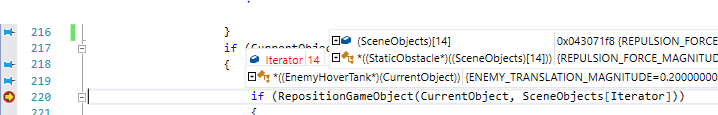
This method simply adds the (negated) movement direction of this object multiplied by the parsed-in RepulsionMagnitude, to cause the ControlledObject to bounce back appropriately, away from this StaticObstacle:

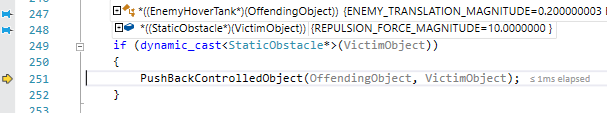
So therefore, this test is successful.

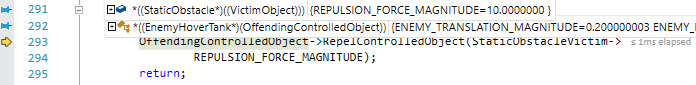
### Test 3

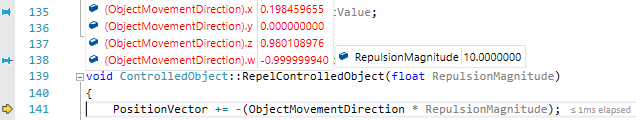
For this test, the Enemy’s hover-tank must be close enough to a StaticRock, as shown here:

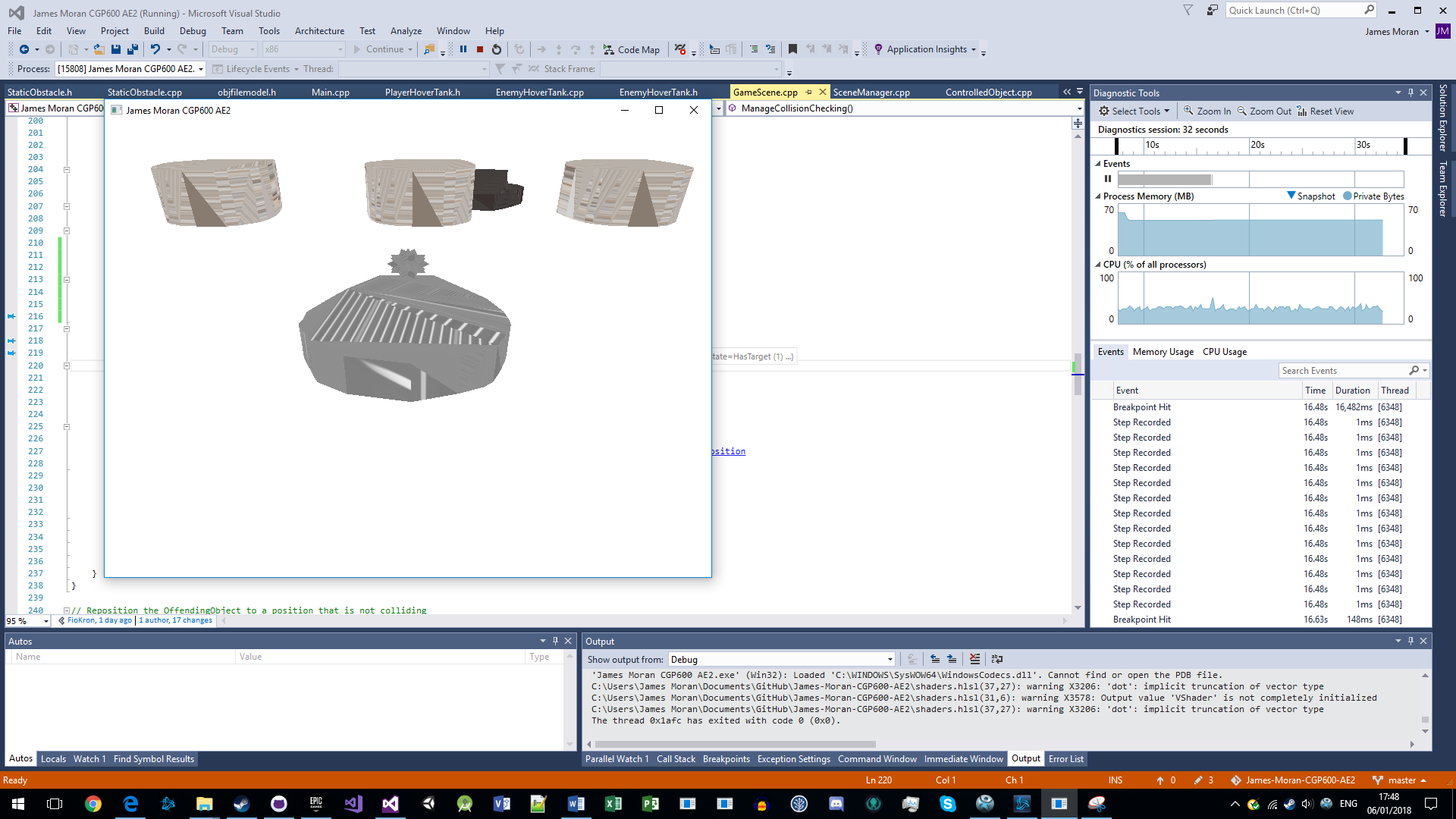
 When they collide with the rock, the breakpoint as shown on the next page is triggered:

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

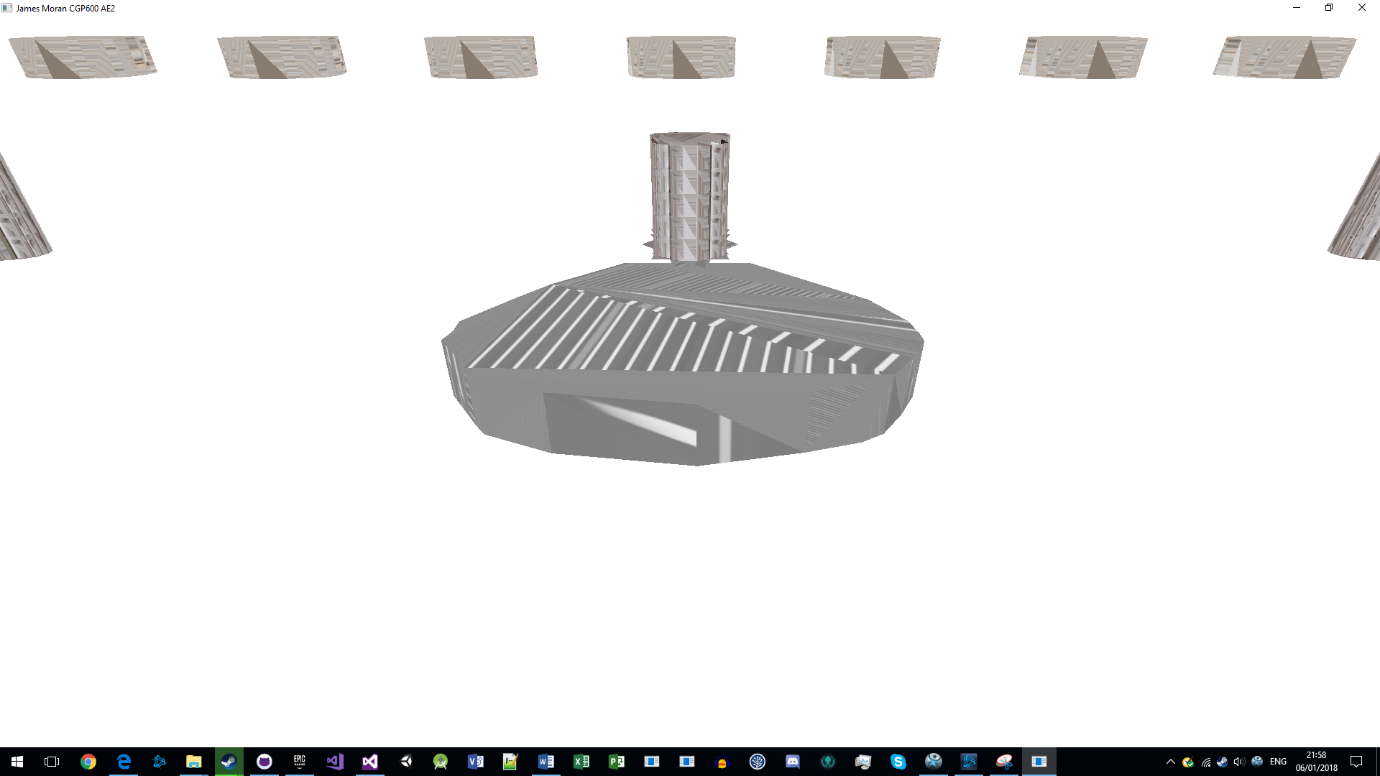
Which then calls the PushBackControlledObject() method:

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

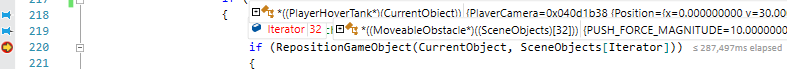
Which prevents the Enemy’s hover-tank from getting near to the Player through this StaticRock:

Therefore, this test is successful.

### Test 4

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

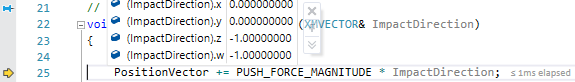
On the next page is the breakpoint triggered once again, in the GameScene’s ManageCollisionChecking() method:

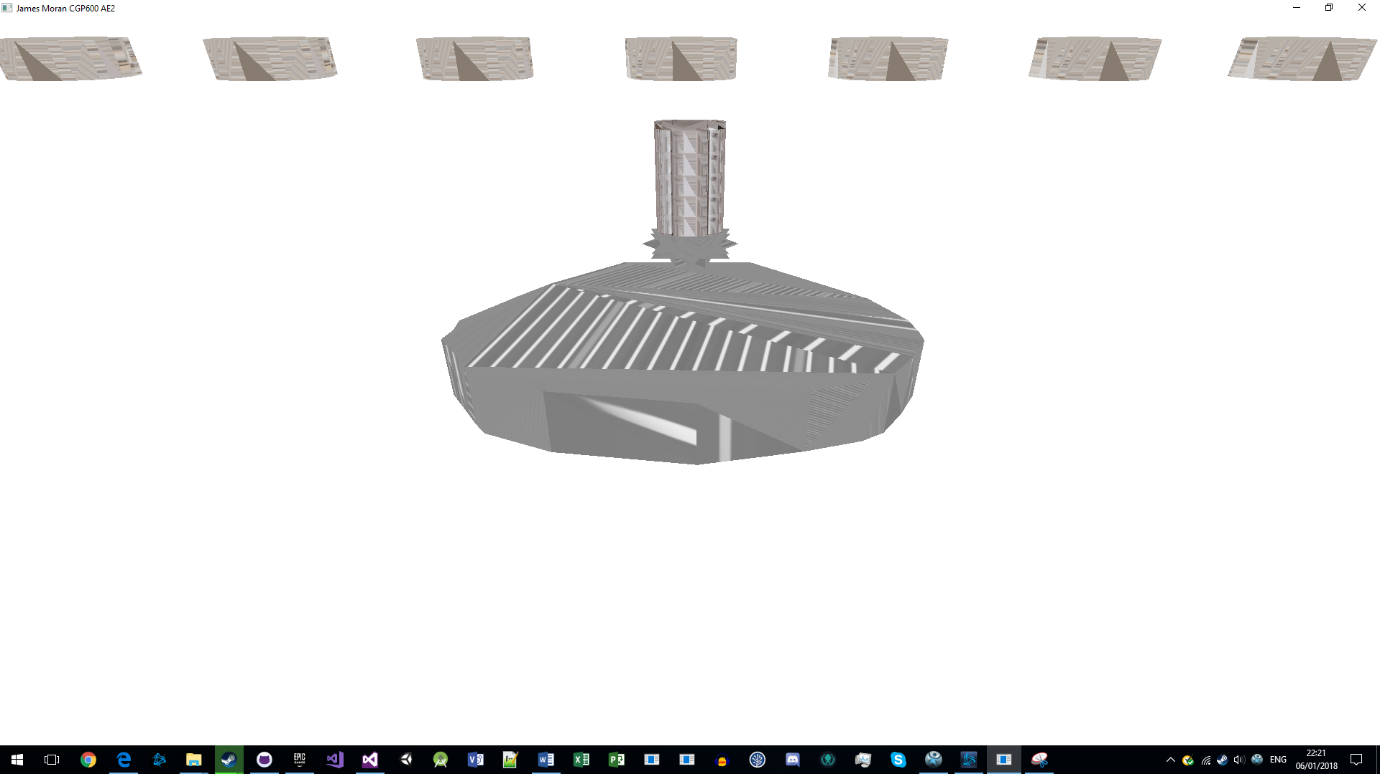
Which then calls RepositionObject():

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



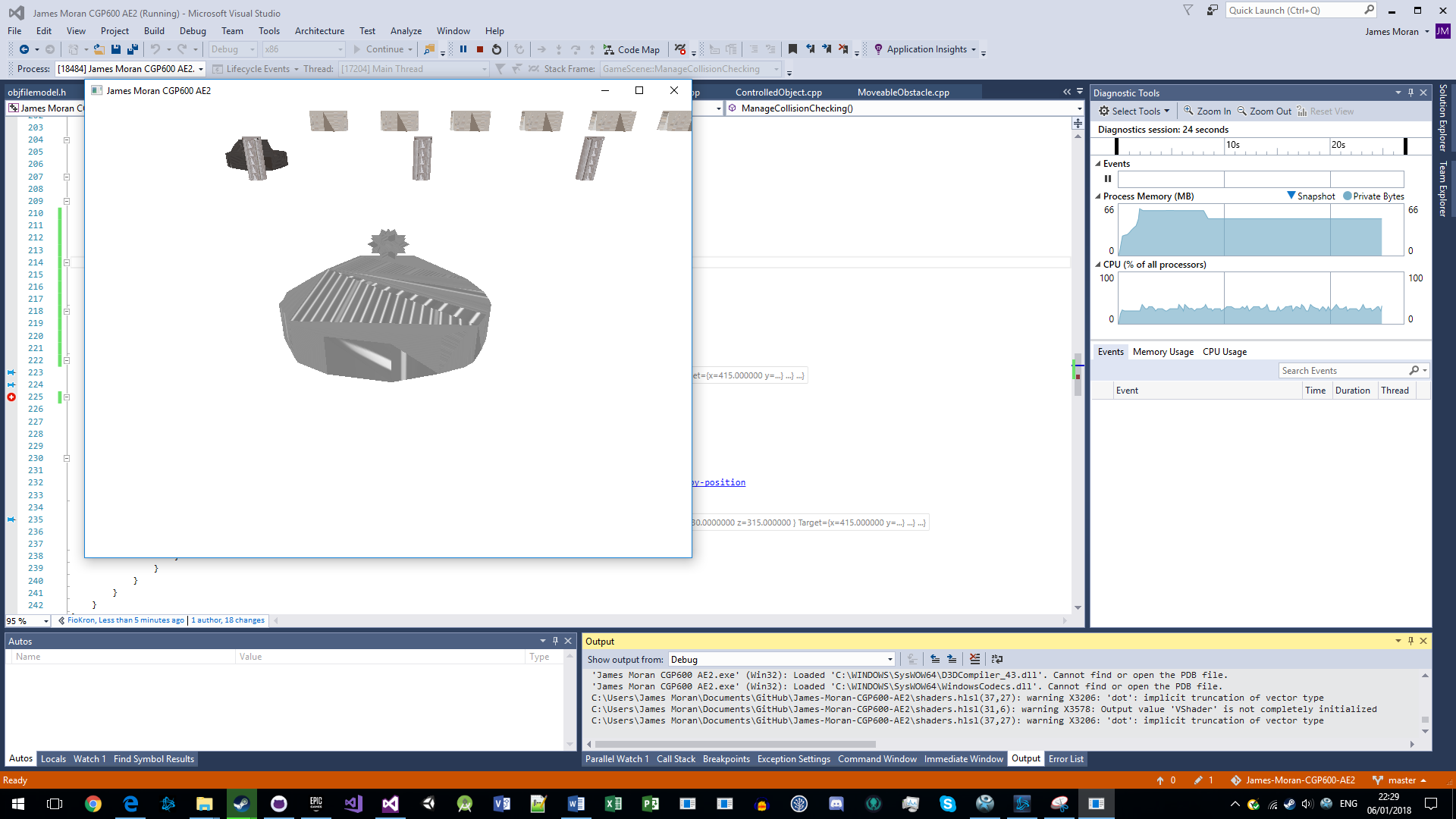
Which then calls the OnImpact() method of the MoveableObstacle (parsing in the ControlledObject’s MovementDirection):

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.

### Test 5

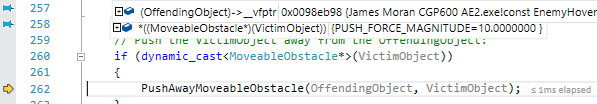
For this test, an EnemyHoverTank must be near to a MoveableObstacle:



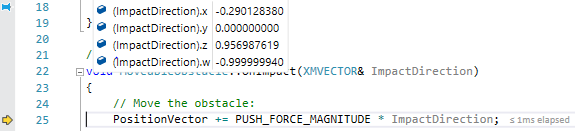
Then the same breakpoint (with different conditions) is triggered when they collide with this WoodenBarrel (as shown on the next page):

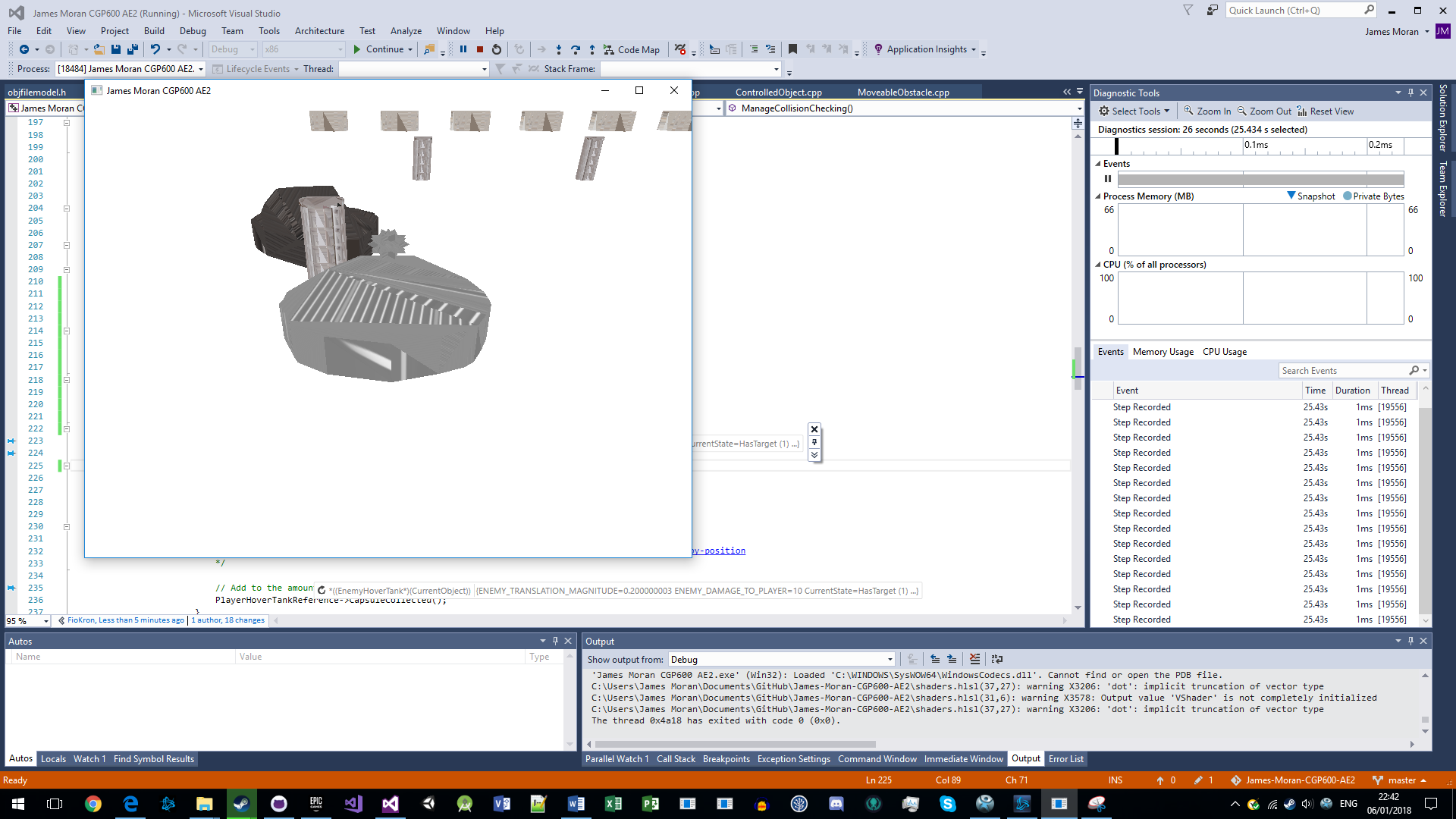


Which then calls RepositionObject():

Which then calls PushAwayMoveableObstacle():

Which then calls OnImpact():

Which then results in movement of the Wooden-Barrel:

Therefore, this test is successful.

### Test 6

# 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. Master Kenneth, 2014, DirectX Leak Debugging [viewed on the 07/12/2017]. Available from: <http://masterkenth.com/directx-leak-debugging/>
5. 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/
6. 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>
7. Allen Sherrod and Wendy Jones, 2012. *Beginning DirectX® 11 Game Programming.* Boston, MA: Course Technology PTR
8. John McGrath and James Moran, 2017. *CGP600 Advanced Games Programming AE1 – Group Project.* Southampton: Southampton Solent University.
9. © 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>
10. 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)
11. 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)