Notes on the Conversion of the Tyche Simulation Engine from Visual Basic 6.0 to Visual C#.NET

Tyche 3.0 Development Project

Terry Restoule
Contractor, LeverageTek IT Solutions

136 Lewis Street, Suite 1
Ottawa, ON K2P 0S7
Contract Project Manager: Mark Provenzano, 613-680-2933
PWGSC Contract Number: W7714-3810
CSA: Cheryl Eisler, 613-947-9796

DRDC CORA CR 2013–156
September 2013

Defence R&D Canada
Centre for Operational Research and Analysis

Maritime Operational Research Team
Maritime and Air Systems Section

National Defence Défense nationale

Canada

Notes on the Conversion of the Tyche Simulation Engine from Visual Basic 6.0 to Visual C#.NET

Tyche 3.0 Development Project

Terry Restoule
Contractor

Prepared By:
Terry Restoule
LeverageTek IT Solutions
136 Lewis Street, Suite 1
Ottawa, ON
K2P 0S7
Contractor's Document Number: N/A
Contract Project Manager: Mark Provenzano, 613-680-2933
PWGSC Contract Number: W7714-3810
CSA: Cheryl Eisler, 613-947-9796

Defence Research and Development Canada – CORA

Contract Report
DRDC CORA CR 2013-156
September 2013

IMPORTANT INFORMATIVE STATEMENTS

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

Template in use: template-july2013-eng_V.03.01.dot

Notes on the Conversion of the Tyche Simulation Engine from Visual Basic 6.0 to Visual C#.NET

Tyche 3.0 Development Project

Prepared By:
Terry Restoule
LeverageTek IT Solutions
136 Lewis Street Suite 1, Ottawa, ON, K2P 0S7
Contract Number: W7714-3810
Contract Scientific Authority: Cheryl Eisler, Maritime Operational Research Team, 613-947-9796

Version 4.0
Latest Revision
October 1, 2013

RECORD OF AMENDMENTS

Amendment No.	Entered By	Revisions	Date
1	Terry Restoule	Original Version	Mar 19, 2013
2	Terry Restoule	Response to comments	May 15,2013
3	Terry Restoule	Response to comments	July 31, 2013
4	Cheryl Eisler	Final publication edits	October 1, 2013

ABSTRACT

The objective of this work was to redevelop the Tyche Simulation Engine software using the integrated development environment (Visual C#.NET) recommended in a preceding technical evaluation. Data structures were transformed to reflect the performance results found during the technical evaluation. The software was directly translated to allow the new system to be easily tested in parallel with the original Visual Basic code. Once the new software produced output consistent with the Visual Basic version, the new code was then restructured to improve maintainability. The redeveloped code ran approximately twice as fast as the Visual Basic version during testing.

RÉSUMÉ

W Les travaux effectués avaient pour but la refonte du moteur logiciel de simulation Tyche à l'aide de l'environnement de développement intégré recommandé dans une évaluation technique précédente, c'est-à-dire Visual C#.NET. Nous avons revu les structures de données en fonction des performances observées dans l'évaluation technique. Le logiciel a été porté directement, pour qu'on puisse comparer facilement le nouveau système en parallèle avec le code Visual Basic (VB) d'origine. Une fois que le nouveau logiciel a produit des résultats identiques à ceux de la version VB, nous avons restructuré le code C# pour en simplifier les modifications ultérieures. Après redéveloppement, les essais ont révélé que ce code roule à peu près deux fois plus vite que la version VB.

EXECUTIVE SUMMARY

The Tyche Simulation Engine and Graphical User Interface (GUI) programs were developed using Visual Basic 6.0 (SP6), which is no longer supported by Microsoft. To determine a successor to Visual Basic, a number of commercially available and open source programming languages and software packages were evaluated. Microsoft Visual C#.NET (C#) was selected to be used in the redevelopment effort.

During the technical evaluation, it was found that the class and collection object approach with respect to the internal data had performance drawbacks and it was decided to store the data in structures (structs), dynamic arrays, and lists as appropriate.

The code was translated as directly as possible to allow the new system to be easily tested in parallel with the older one. The new system would be considered functional when it produced output identical to the old one on a specific set of test cases.

Once the new system was found to be functional, the performance needed to be improved. This was done by removing extraneous table lookups. It was also decided that the code would be restructured to remove unnecessary local variables, and to directly reference and update the static array structures as needed. The restructuring was also intended to improve maintainability of the code.

This report describes the approach taken toward the redevelopment of the Simulation Engine, the structure of the new application (new class files); the structure of the new data environment (the replacement of the class objects used in the Visual Basic version with the corresponding dynamic arrays in C#); and how it was transformed and the ways in which the code was restructured. Timing data is also presented to show a quantitative performance improvement.

Running test simulations with the same input and parameters, the new C# Simulation Engine software runs in slightly less than half the time of the Visual Basic version.

SOMMAIRE

Le moteur de simulation Tyche et l'interface graphique connexe ont été développés en Visual Basic 6.0 SP6, un environnement pour lequel Microsoft n'offre désormais plus aucun soutien technique. Pour choisir son successeur, de nombreux produits commerciaux, langages de programmation et progiciels en source ouverte ont été évalués. En fin de compte, Microsoft Visual C#.NET (C#) a été jugé l'environnement se prêtant le mieux au redéveloppement prévu.

L'évaluation technique a révélé que la gestion des données internes par classes et collections d'objets entraînait une lenteur indue à l'exécution; le choix s'est donc plutôt porté vers l'utilisation de structures de données, de tableaux dynamiques et de listes, selon la méthode la mieux appropriée.

Le logiciel a été porté d'un langage à l'autre aussi directement que possible, pour qu'on puisse comparer facilement le nouveau système en parallèle avec le code d'origine. On a décidé de ne juger le nouveau système fonctionnel qu'une fois qu'il produisait à partir d'une série précise de données d'essai des résultats identiques à la version précédente.

Une fois fonctionnel, il a ensuite fallu en optimiser l'exécution, tout d'abord en éliminant les recherches inutiles dans les tableaux. Nous avons aussi restructuré le code pour éliminer les variables inutiles et pointer directement aux structures et tableaux de données statiques, tant pour la lecture que l'écriture. Cette restructuration visait aussi à simplifier les modifications ultérieures au code.

Le rapport décrit la démarche adoptée pour redévelopper le moteur de simulation, la structure de la nouvelle application (les nouvelles classes), les nouvelles structures de données internes (le remplacement des objets utilisés en VB par les tableaux dynamiques de C#), et de façon générale comment le code a été transformé et restructuré. Le rapport présente aussi les résultats des essais de performance, afin de chiffrer l'optimisation du rendement obtenue.

Dans les simulations d'essais effectuées avec les mêmes données et paramètres, le nouveau moteur de simulation C# a exécuté les choses un peu plus de deux fois plus vite que la version Visual Basic.

TABLE OF CONTENTS

1. INTRODUCTION 7
2. REDEVELOPMENT METHODOLOGY 7
3. APPLICATION ARCHITECTURE 9
4. THE TRANSFORMED DATA 15
5. CODE OPTIMIZATION AND RESTRUCTURING 18
6. THE TEST PLATFORM 19
7. TESTING AND TEST RESULTS 19
8. CONCLUSIONS 20

Appendix A - Test Logs 21

1. INTRODUCTION

The Tyche Simulation Engine and Graphical User Interface (GUI) programs were developed using Visual Basic 6 (SP6), which is no longer supported by Microsoft. A performance-based technical language selection process[1] recommended that Microsoft Visual C#.NET (C#) should be used to redevelop the software.

The technical evaluation also demonstrated that performance improvements could be obtained by moving away from dynamic class and collection objects in favour of structures and dynamic arrays. These changes were incorporated into the new system.

Once a version of the new simulation engine successfully could produce output that agrees with the VB6 version, the code would be re-structured for better clarity and maintainability.

2. REDEVELOPMENT METHODOLOGY

Due to the complexity of the model's internal algorithms, it was decided to translate the code as directly as possible. Direct translation would allow the new version of the application to be more easily tested side by side with the old version, and would simplify the tracing and correction of problems. It would also minimize the chances of misinterpretation of the previous version's processing code.

The elimination of collections of class objects meant that class files similar to the .cls files in the Visual Basic version would not have to be created. In their place, a new TycheData class was created containing the data structures for the various dynamic arrays and lists needed. A structure (struct) was defined for each of the class objects to be replaced. These structures contained fields based on the properties of their corresponding class objects. Where a class object contained a collection of objects in the past, a new array field of the appropriate structure type was included. The new TycheData class also contained definitions for all publicly shared variables; along with past class object method code that was still required.

Most of the C# code methods retained the names and calling structures of their corresponding functions and subroutines in the VB6 version. Variables in the C# version were defined with the same names as the class object variables used locally within the Visual Basic code. These variables were defined to be of the replacement structure defined in the TycheData class. For example, in Visual Basic, a local variable "thisAsset" would be declared as:

```
thisAsset As clsAsset
```

In C#, the corresponding variable would be declared as:

```
TycheData.Asset thisAsset
```

This approach allowed the new version to retain the code structure of the previous version to a large extent.

[1] Eisler, C. (2012), Tyche 3.0 Development: Comparison of Development Environments for a Monte Carlo Discrete Event Simulation (MCDES), (DRDC CORA TM 2012-231), Defence Research and Development Canada.

The representation of data in structures rather than classes presented its own challenges. One of the consequences of using a structure was that field values on a record could not be passed by reference. Whenever a field value had to be passed into a routine and updated, a variable had to be declared and then loaded with the data from the structure. The new variable (or field value) could be passed by reference. Once the modified value was returned from the method, the global structure item would have to be updated with the contents of the local variable that had been used.

The code was translated and tested in stages. The code necessary to initialize the processing environment was created first. This included the handling of command line parameters, reading from and writing to the registry, variable initialization and the loading of input data.

The code used to generate the statistical output was translated next because the data analysis was a self-contained process. In comparison with the main iteration code, the statistical analysis code was relatively straightforward, and much was learned about the mechanics of how best to translate the code. Once the statistical analysis code was ready, it was tested by having it interpret simulation output data created by the Visual Basic application to produce comparable statistics. During this testing, errors were found in the Visual Basic code and the C# code was corrected to address these errors.

The main iteration processing code was the last code to be translated. This allowed it to take advantage of all that had been learned in the translation of the other code. In order to verify the accuracy of the code, the new version of the engine had to be able to create output that was identical to the output produced by the Visual Basic version running the same input data with the same run parameters.

As with the statistical analysis, bugs in the Visual Basic code were found from time to time. Since the output from the C# application had to be identical in order to verify processing accuracy, the C# code had to be "broken" to match the output of the old code. When these situations occurred, the correct code was left in as comments and further comments were added to the code so that the C# code could be easily fixed at a later date (i.e., in the next task authorization).

3. APPLICATION ARCHITECTURE

The new C# application's code exists across 12 code classes. The following table identifies each of these class files and describes contents relative to the Visual Basic version.

Table 1: Class Files with Descriptions.

Class File Name	Description
DataAnalysis.cs	This class contains the logic necessary to produce the run statistics analysis (modDataAnalysis.bas). It also contains the data structures equivalent to clsDataAnalysisAsset.cls, clsDataAnalysisLevel.cls and clsDataAnalysisPhase.cls.
Logging.cs	This class produces all of the messages written to the application log, the XML file and the console screen (modLogging.bas).
MathProcs.cs	This class is equivalent to modMathFunction.bas. The routines used in the Visual Basic version to determine Maximum and Minimum values were dropped in favour of the C# intrinsic functions.
RandomProcs.cs	This class is equivalent to modRandom.bas.
RegAssets.cs	This class is equivalent to modAssetsRegistration.bas.
RunAdmin.cs	Contains the code from modSubmitJob.bas and modSystem.bas.
RunFileIO.cs	This class is equivalent to modFileIO.bas with the following exceptions: • The CountSeparator routine was no longer needed and was removed; • The family of AddElementToArray overloads were added
SelAssets.cs	This class is equivalent to modAssetsSelection.bas
SimSetup.cs	Contains the code from modSimulationSetup.bas and modSimulationRun.bas
SimXML.cs	This class is equivalent to modSimulationInstance.bas.
TycheData.cs	See section 4 for details.
TycheMain.cs	Contains the code from modMain.bas and modCommandLine.bas.

Due to the direct translation approach used, most of the classes mentioned in the table above closely resemble their Visual Basic predecessors. The notable exceptions are SimSetup.cs, SelAssets.cs and RegAssets.cs. These classes contain methods based on long and repetitive Visual Basic subroutines that were restructured to simplify the code and enhance its maintainability.

In SimSetup.cs, the GenerateEvents method which is analogous to the GenerateEvents subroutine in the Visual Basic modSimulationSetup.bas, was a single large routine. Its processing was simplified by moving portions of its code to asset-based and scenario-based versions of the GenScheduled and GenRandom methods. These methods, in turn, call other new, smaller methods intended to make the code more manageable. The following call diagram illustrates how these methods are inter-related.

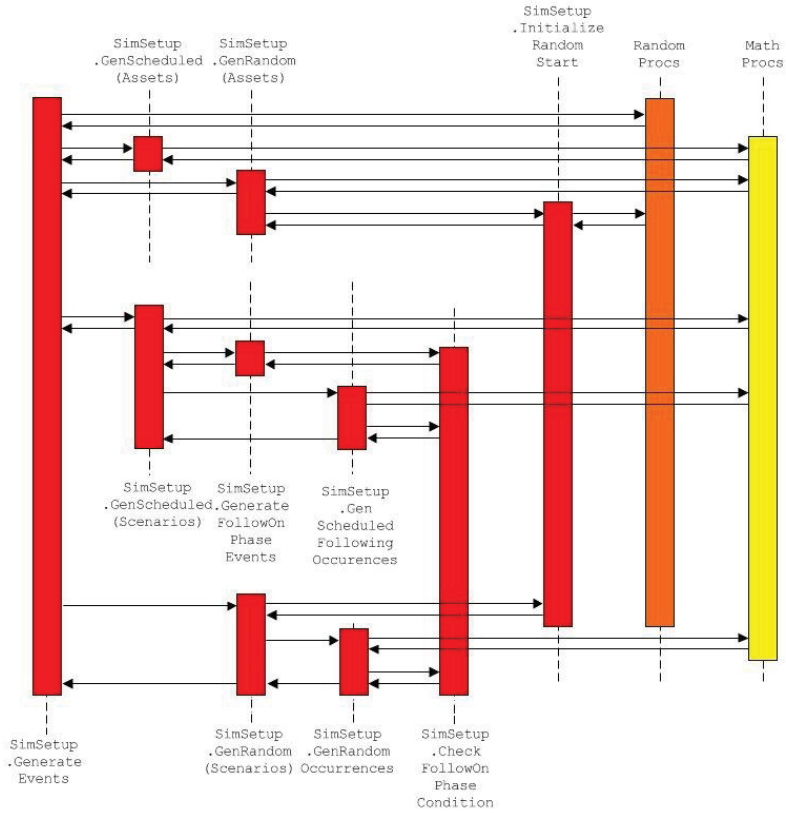

Figure 1: GenerateEvents Call Diagram[2]

Similarly, the ScheduleAssets method is equivalent to the ScheduleAssets subroutine in the Visual Basic module modSimulationRun.bas. Its processing was also spread across several new methods to improve code manageability. These new methods included UpdateAllAssetActivity, ProcessScenarioEvent and two methods addressing asset event processing depending upon whether or not the asset has future events. The following call diagram illustrates how the ScheduleAssets method is now structured.

[2] The author would like to thank Cheryl Eisler for the initial development of the call diagrams and the entity relationship diagram in this report.

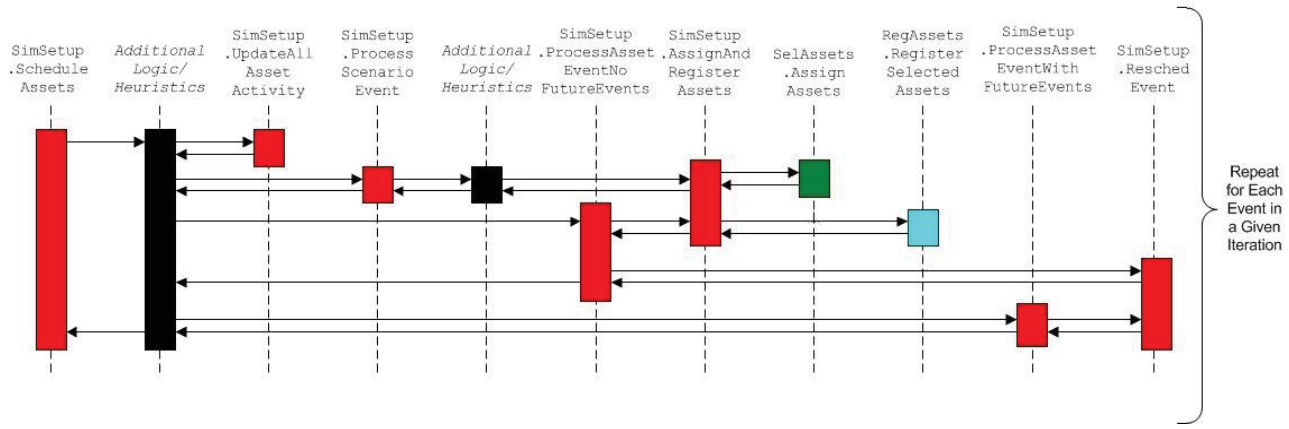

Figure 2: ScheduleAssets Call Diagram

In SelAsset.cs, the AssignAssets method is analogous to the AssignAssets subroutine in the Visual Basic module modAssetsSelection.bas. In the interest of maintainability, the processing in the AssignAssets method was broken into a set of helper methods. The new methods were created in such a way as to maintain the original structure of the AssignAssets routine.

At the beginning of the AssignAssets method, array initialization is performed by two new methods. There are also separate initialization methods for scenario and asset events.

In the main processing loop, methods are called to determine which assets are available for selection and to evaluate those assets for selection. New methods were also created to contain other distinct portions of processing. These methods initialize processing at a new layer, check scoring thresholds, setup for reprocessing at a previous layer, check for redundant processing, remove unused external assets and verify essential assets. New methods also were created to set unmatched capability arrays and to store capabilities met by the assets.

Several methods were created to encapsulate processing that previously had been duplicated throughout the routine. These include methods to add to capability demand, to abandon selected assets and to prepare for a new asset search.

The call diagram on the following page illustrates how the asset assignment process is now structured.

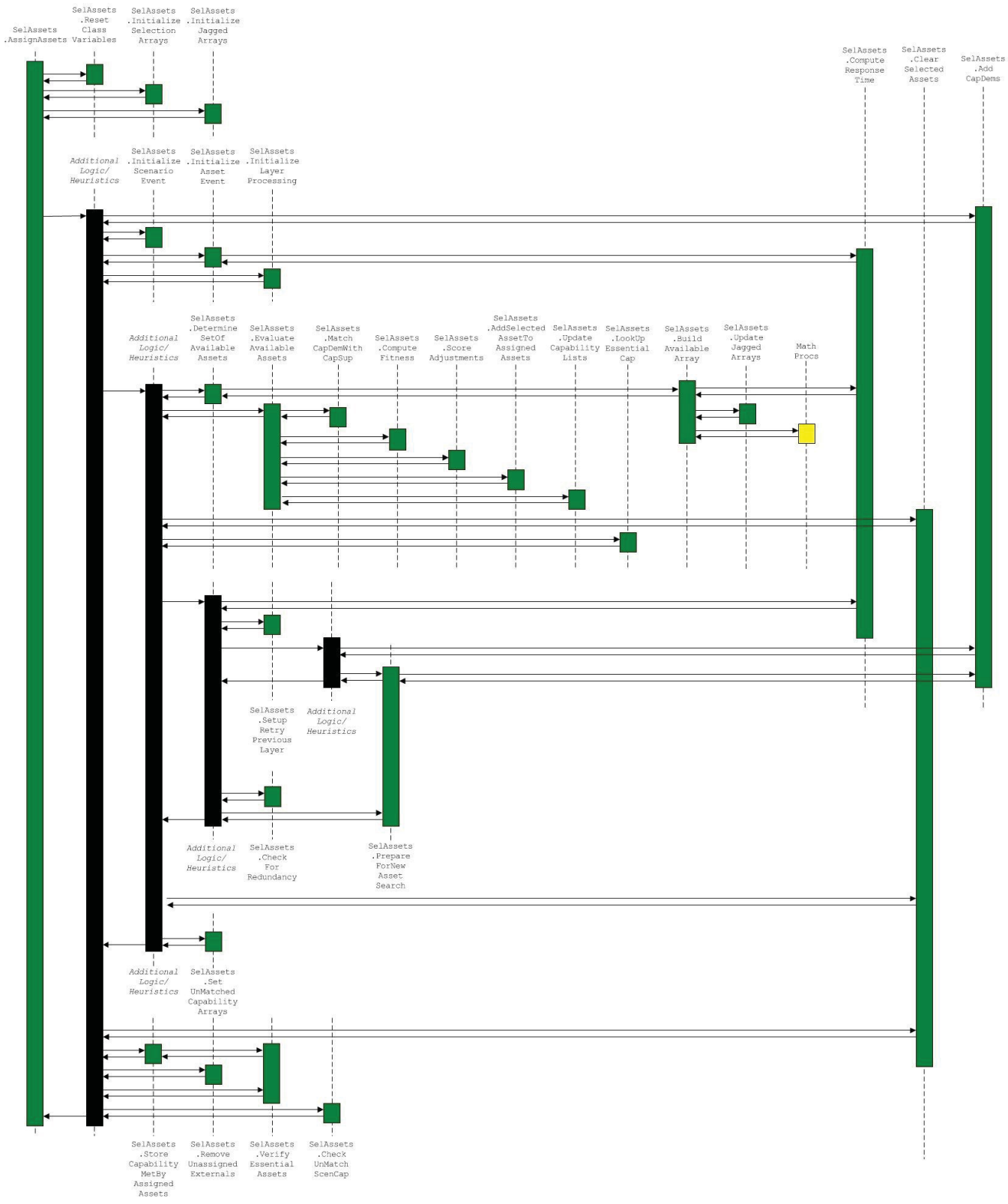

Figure 3: AssignAssets Call Diagram

In RegAssets.cs, the RegisterSelectedAssets method is analogous to the RegisterSelectedAssets subroutine in the Visual Basic module modAssetsRegistration.bas. As with the methods mentioned in SimSetup.cs and SelAssets.cs, the RegisterSelectedAssets method was restructured in the interest of maintainability.

A set of new methods were created to encapsulate discrete portions of processing while allowing the original structure of the RegisterSelectedAssets method to be preserved. The following is a list of the major new methods:

- FindReplacementAssets
- AddAssetToCheckedAssets
- AddReplacementAssetToAssignedAssets
- RegisterAssetToEssentials
- UpdateEssentialAssetDates
- RemoveCurrentEventAssetsFromTheatre
- UpdateReplacedAssetAssignment
- UpdateBumpedAssetFutureEventsCurrentActivity
- CheckFollowOnEvents

The call diagram on the following page illustrates how RegisterSelectedAssets is now structured.

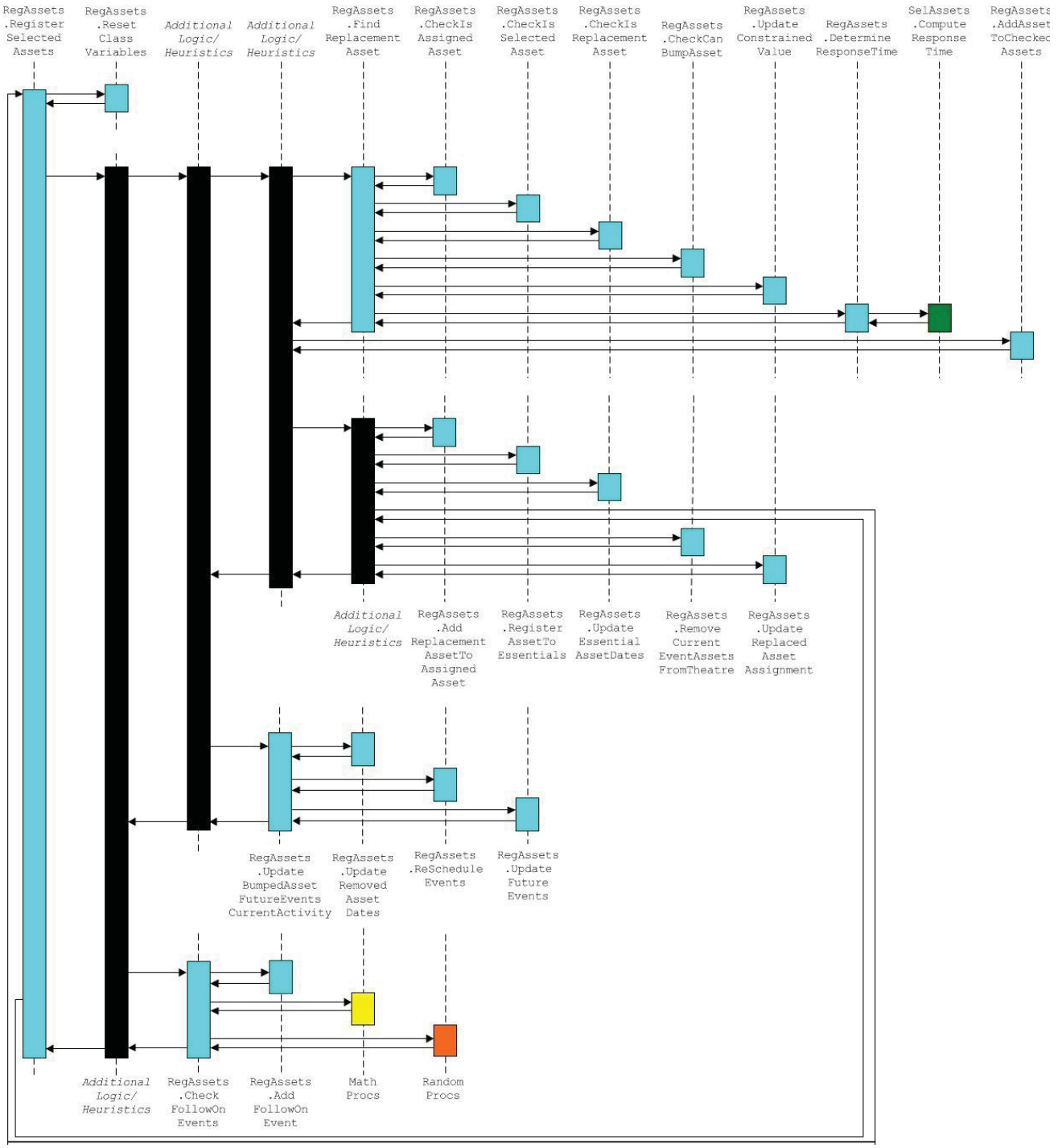

Figure 4: RegisterSelectedAssets Call Diagram

4. THE TRANSFORMED DATA

Where collections of class objects were used in the Visual Basic version of the simulation engine, dynamic arrays of structures are now used in the C# version. Each of the dynamic arrays was based on structures derived from the properties of their respective Visual Basic class objects. A set of public static arrays of each of these structures were then created in a central data repository, populated and manipulated throughout the course of the simulation processing. A single method "GetElementIndex" was created to retrieve the index of the record required from a named array. There were other functions for array manipulation, as well as a number of new functions that replaced the class methods in Visual Basic.

The table on the following page describes how the C# structures map back to their Visual Basic counterparts. The new global arrays are also identified.

Not all of the arrays and collections used in the Visual Basic version were defined globally. In the following table, the Visual Basic structures are defined globally except where indicated.

Refer to the notes following the table for further information regarding the implementation in C#.

Table 2: Class Object Mapping.

Visual Basic		**C#**	
Class Object	**Collection**	**Structure Definition**	**Static Array(s)**
clsAsset.cls	arrAssets, arrExternalAssets	Asset	arrAssets, arrExternalAssets
clsAssignedAsset.cls	arrAssignedAssets	AssetAssignment	arrAssetAssignments
clsAssetType.cls	colAssetTypes	AssetType	arrAssetTypes
clsBase.cls	colBases	Base	arrBases
clsBaseDistance.cls	colBaseDistances	BaseDistance	arrBaseDistances
clsBumpData.cls	colBumpdata (in clsAssetType)	BumpData	arrBumpData
clsCapability.cls	colCapabilities	Capability	arrCapabilities
clsCapDem.cls	colCapDems (in clsLevel and clsPhase)	CapDem	arrCapDems
clsCapSup.cls	colCapSups (in clsLevel)	CapSup	arrCapSups
clsConstraint.cls	colConstrainedValues (in clsAsset), colConstraints (in clsLevel)	Constraint	Constraints (in Level)
clsDataAnalysisAsset.cls	arrAssets (in modDataAnalysis)	DA_Asset	arrAssets
clsDataAnalysisLevel.cls	arrLevels (in clsDataAnalysisAsset)	DA_Level	arrLevels
clsDataAnalysisPhase.cls	arrPhases (in modDataAnalysis)	DA_Phase	arrPhases
clsDistance.cls	colDistances, colBaseDistances (in	Distance	arrDistances

Visual Basic		C#	
Class Object	**Collection**	**Structure Definition**	**Static Array(s)**
	clsBase) colDistances (in clsTheatre)		
clsEvent.cls	colEvents (in modSimulationSetup)	Event	lstEvents
clsFleet.cls	colFleets	Fleet	arrFleets
clsFleetMember.cls	colFleetMembers (in clsFleet)	Member	arrMembers
clsLevel.cls	Levels (in clsAssetType), Levels (in clsMultiLevelConstraint)	Level	arrLevels
clsLevelHistory.cls	WholeHistory (in clsAsset)	LevelHistory	arrLevelHistories
clsMultiLevelConstraint.cls	Constraints (in clsAssetType)	MultiLevelConstraint	MultilevelConstraints (Dictionary in Asset), MultilevelConstraints (in AssetType)
clsPhase.cls	Phases (in clsScenario)	Phase	arrPhases
clsScenario.cls	colScenarios	Scenario	arrScenarios
clsScenarioType.cls	colScenarioTypes	ScenarioType	arrScenarioTypes
clsScoringCriterion.cls	ScoringCriteria (in clsLevel), ScoringCriteria (in clsPhase)	ScoringCriterion	arrScoringCriteria
clsTheatre.cls	colTheatres	Theatre	arrTheatres

1. The arrays (arrAssets, arrLevels, arrPhases) based on the structures DA_Asset, DA_Level and DA_Phase are defined locally within the DataAnalysis.cs and are NOT available globally.
2. lstEvents is a C# intrinsic List structure and not an array. The List structure was used so that the events could be efficiently maintained in start date order.

On the next page is the final Entity/Relationship Diagram that illustrates how the data is used by the Simulation Engine. The entities (boxes in the diagram) represent the various structures defined in the program. The relationships (lines in the diagram) identify how the various entities work together. The connectors at each entity identify whether the relationship is one-to-one or one-to-many.

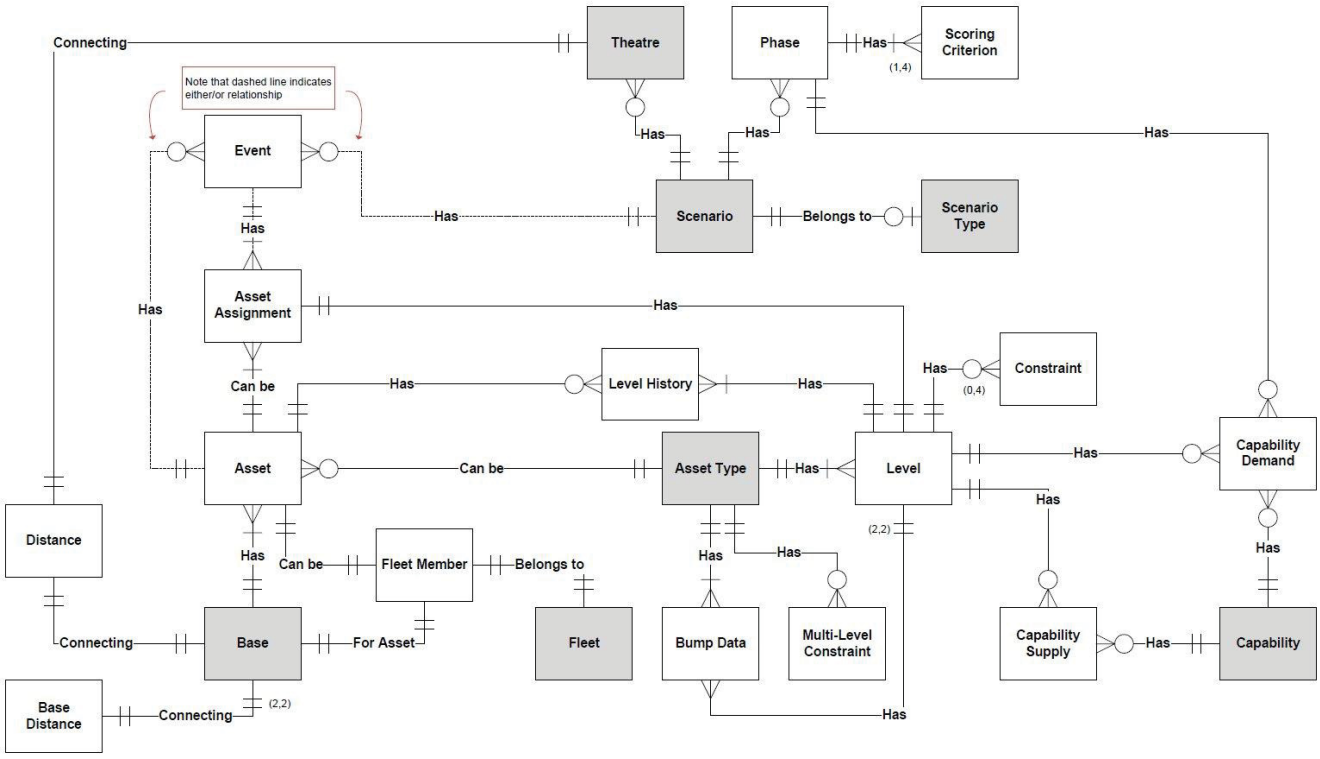

Figure 5: Tyche Simulation Engine Entity/Relationship Diagram

5. CODE OPTIMIZATION AND RESTRUCTURING

Once the C# version was largely functional, it became clear that the performance could be improved. That improvement was accomplished through the following strategies:

1. Removal of extraneous table look-ups.
2. The replacement of temporary working variables with direct references and updates to the global arrays.
3. Improved code re-use through shorter methods, eliminating duplication and making future upgrades easier.

In addition, the calling of many methods was simplified through variable scoping. Many variables that had been defined local to a method and passed to other methods through parameters were redefined as global within the class. This re-scoping necessitated the creation of a new method called ResetClassVariables that ensured that the class level variables were reinitialized properly for the processing of each event. A version of this method was added to both SelAssets.cs and RegAssets.cs.

The classes central to the processing of the simulation (SimSetup.cs, SelAssets.cs, RegAssets.cs) were restructured to improve the maintainability of the code.

- In SimSetup.cs, the GenerateEvents and ScheduleAssets methods was simplified by splitting the code into sets of helper methods.

- In SelAssets.cs, the AssignAssets method was similarly simplified through the creation of a set of helper methods. The general structure of the original method was preserved. The method BuildAvailableArray was also simplified.

- The RegisterSelectedAssets method in RegAssets.cs was simplified by the creation of a set of helper methods. The general structure of the original method was preserved.

6. THE TEST PLATFORM

Final tests runs for all of the programs were performed on an Acer Aspire M3910 with a 3.2 GHz Intel Core i5-650 CPU (4 MB L3 cache, 3.20 GHz, DDR3 1333 MHz) with the Intel H57 Express Chipset, and 6 GB of RAM.

The test computer ran Windows 7 Home Premium (64 bit), Service Pack 1.

7. TESTING AND RESULTS

Table 3 summarizes the results of the test runs of the simulation performing 1000 iterations of a seven year simulation period. Two versions of the C# program were used. The optimized version was built with the "Optimize code" option checked on the "Build" tab of the project's Properties. This option was unchecked when the "Not Optimized" version was built.

Table 3: Average iteration run time results.

Platform	Test Description	Results
Visual Basic 6	Full Simulation Run Average Time/Iteration Statistics Generation	2 hours 42 minutes 28 seconds 9.748 seconds 42 seconds
Visual C# (Not Optimized)	Full Simulation Run Average Time/Iteration Statistics Generation	1 hour 17 minutes 42 seconds 4.622 seconds 21 seconds
Visual C# (Optimized)	Full Simulation Run Average Time/Iteration Statistics Generation	1 hour 13 minutes 18 seconds 4.398 seconds 20 seconds

The optimized version of the C# program runs the simulation in 45% of the time taken by the Visual Basic version.

It is recommended that the "Optimize code" option should be checked on the "Build" tab of the project's Properties because the optimization does produce slightly faster code.

8. CONCLUSIONS

Though the new C# version of the simulation engine showed a significant improvement in processing speed over the Visual Basic-based one, the speed improvement was not as dramatic as the results in the technology evaluation predicted. The better speed in the evaluation program may have been due to the simplicity of the structures used. The structures used in the simulation engine are larger and more complex.

Whenever an array is resized, a new copy with altered size is allocated and then the data in the current array is then copied to the new array. For large arrays, this overhead can be significant. In the simulation engine, several of the structures that are the basis of arrays contain fields which are themselves arrays. When any of these array fields are resized, the overhead is compounded. Further investigation will have to be conducted to determine what the actual overhead of array resizing is and how best to address it.

Moving forward, a number of issues with the code must be addressed. They include:

- During the testing of the C# version a few bugs were found in the Visual Basic code. Rather than fix the code in the old system, the C# code was modified so that its output would agree with the Visual Basic output. These changes must now be removed.
- "Base Distances" have been kept separate from "Distances". This isn't necessary and Base Distances should be handled simply as distances.
- The console window used for the main simulation program and the output zipping operation should be removed and any lingering issues with Dashboard interaction must be resolved.

APPENDIX A - TEST LOGS

When a simulation is run, an XML file containing the run parameters and log messages is produced.

The following are the log results for each of the test runs used in Section 7. The date and time stamp on the log messages enable the user to calculate the total time required to complete the simulation run.

Visual Basic

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This is an Extensible Markup Language file (XML). This template is to be used to run
an unattended simulation on a Tyche input file (tyi). Execute Tyche with the path to this
XML file as the argument. Do not make changes above this line. -->
<execution command="run">
        <input-file>c:\tyche\test data\dummy datav17.tyi</input-file>
        <fleet>1a</fleet>               <!-- Choose only one fleet for the simulation. (case sensitive) -->
        <iterations>1000</iterations> <!-- Iterations must be a positive integer. -->
        <years>7</years>                <!-- Years must be a positive integer between 1 and 80 inclusive. -->
        <!--<seed>-2147</seed>-->  <!-- OPTIONAL. Override the simulation random seed value. -->
        <scenario-types>
                <!-- Choose at least one scenario type. (case sensitive) -->
                <scenario-type>3c4d</scenario-type>
                <scenario-type>1a2b</scenario-type>
        </scenario-types>
        <!-- OPTIONAL. Specify a name for the output directory.
        NOTE: Existing output files will be overwritten. Omit this tag to generate the
        result directory automatically as a subdirectory of the input file's location. -->
        <result-directory>c:\tyche\test data\1a 130429 02H59S21</result-directory>
        <generate>
                <apply-specialized-lift-capability-rules>True</apply-specialized-lift-capability-rules>
                <!-- Choose the types of statistics (asset, scenario and/or capability) to generate. -->
                <asset-statistics>True</asset-statistics>
                <scenario-statistics>True</scenario-statistics>
                <capability-statistics>True</capability-statistics>
                <compute-risk>True</compute-risk>
        </generate>
        <log date="2013-04-29" time="02:59:21">Saved simulation parameters to new XML file</log>
        <log date="2013-04-29" time="02:59:22">Simulation Starting: My unique ID is Node USSW
792</log>
        <log date="2013-04-29" time="05:41:50">Simulation Completed</log>
        <log date="2013-04-29" time="05:41:50">Statistics Generation Starting</log>
        <log date="2013-04-29" time="05:41:50">Built the assets used for collection and calculation</log>
        <log date="2013-04-29" time="05:41:51">Built the phases used in collection and calculation</log>
        <log date="2013-04-29" time="05:41:53">Built the data structure used in collection and calculation
of risk</log>
```

```
        <log date="2013-04-29" time="05:42:29">Collected data from output file</log>
        <log date="2013-04-29" time="05:42:29">Completed Asset Statistics Generation</log>
        <log date="2013-04-29" time="05:42:29">Completed Scenario Statistics Generation</log>
        <log date="2013-04-29" time="05:42:29">Completed Capability Statistics Generation</log>
        <log date="2013-04-29" time="05:42:32">Completed Risk Spreadsheet Update</log>
        <log date="2013-04-29" time="05:42:32">Statistics Generation Completed</log>
        <log date="2013-04-29" time="05:42:39">TYO file successfully zipped</log>
        <log date="2013-04-29" time="05:42:39">Run Finished</log>
</execution>
```

Visual C#.Net (Not Optimized)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This is an Extensible Markup Language file (XML). This template is to be used to run
an unattended simulation on a Tyche input file (tyi). Execute Tyche with the path to this
XML file as the argument. Do not make changes above this line. -->
<execution command="run">
        <input-file>c:\tyche\test data\dummy datav17.tyi</input-file>
        <fleet>1a</fleet>               <!-- Choose only one fleet for the simulation. (case sensitive) -->
        <iterations>1000</iterations> <!-- Iterations must be a positive integer. -->
        <years>7</years>                <!-- Years must be a positive integer between 1 and 80 inclusive. -->
        <!--<seed>-2147</seed>-->  <!-- OPTIONAL. Override the simulation random seed value. -->
        <scenario-types>
                <!-- Choose at least one scenario type. (case sensitive) -->
                <scenario-type>3c4d</scenario-type>
                <scenario-type>1a2b</scenario-type>
        </scenario-types>
        <!-- OPTIONAL. Specify a name for the output directory.
        NOTE: Existing output files will be overwritten. Omit this tag to generate the
        result directory automatically as a subdirectory of the input file's location. -->
        <result-directory>c:\tyche\test data\1a</result-directory>
        <generate>
                <apply-specialized-lift-capability-rules>True</apply-specialized-lift-capability-rules>
                <!-- Choose the types of statistics (asset, scenario and/or capability) to generate. -->
                <asset-statistics>True</asset-statistics>
                <scenario-statistics>True</scenario-statistics>
                <capability-statistics>True</capability-statistics>
                <compute-risk>True</compute-risk>
        </generate>
        <log date="2013-03-19" time="18:06:23">Saved simulation parameters to new XML file</log>
        <log date="2013-03-19" time="18:06:37">Simulation Starting: My unique ID is Node IGQX
174</log>
        <log date="2013-03-19" time="19:24:19">Simulation Completed</log>
        <log date="2013-03-19" time="19:24:19">Statistics Generation Starting</log>
        <log date="2013-03-19" time="19:24:19">Built the assets used for collection and calculation</log>
        <log date="2013-03-19" time="19:24:19">Built the phases used in collection and calculation</log>
        <log date="2013-03-19" time="19:24:23">Built the data structure used in collection and calculation
of risk</log>
        <log date="2013-03-19" time="19:24:35">Collected data from output file</log>
        <log date="2013-03-19" time="19:24:35">Completed Asset Statistics Generation</log>
        <log date="2013-03-19" time="19:24:35">Completed Scenario Statistics Generation</log>
        <log date="2013-03-19" time="19:24:35">Completed Capability Statistics Generation</log>
        <log date="2013-03-19" time="19:24:40">Completed Risk Spreadsheet Update</log>
        <log date="2013-03-19" time="19:24:40">Statistics Generation Completed</log>
        <log date="2013-03-19" time="19:24:41">Run Finished</log>
</execution>
```

Visual C#.Net (Optimized)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This is an Extensible Markup Language file (XML). This template is to be used to run
an unattended simulation on a Tyche input file (tyi). Execute Tyche with the path to this
XML file as the argument. Do not make changes above this line. -->
<execution command="run">
        <input-file>c:\tyche\test data\dummy datav17.tyi</input-file>
        <fleet>1a</fleet>               <!-- Choose only one fleet for the simulation. (case sensitive) -->
        <iterations>1000</iterations> <!-- Iterations must be a positive integer. -->
        <years>7</years>                <!-- Years must be a positive integer between 1 and 80 inclusive. -->
        <!--<seed>-2147</seed>-->  <!-- OPTIONAL. Override the simulation random seed value. -->
        <scenario-types>
                <!-- Choose at least one scenario type. (case sensitive) -->
                <scenario-type>3c4d</scenario-type>
                <scenario-type>1a2b</scenario-type>
        </scenario-types>
        <!-- OPTIONAL. Specify a name for the output directory.
        NOTE: Existing output files will be overwritten. Omit this tag to generate the
        result directory automatically as a subdirectory of the input file's location. -->
        <result-directory>c:\tyche\test data\1a 130429 15H44S32</result-directory>
        <generate>
                <apply-specialized-lift-capability-rules>True</apply-specialized-lift-capability-rules>
                <!-- Choose the types of statistics (asset, scenario and/or capability) to generate. -->
                <asset-statistics>True</asset-statistics>
                <scenario-statistics>True</scenario-statistics>
                <capability-statistics>True</capability-statistics>
                <compute-risk>True</compute-risk>
        </generate>
        <log date="2013-04-29" time="15:44:32">Saved simulation parameters to new XML file</log>
        <log date="2013-04-29" time="15:44:33">Simulation Starting: My unique ID is Node PDIC
205</log>
        <log date="2013-04-29" time="16:57:51">Simulation Completed</log>
        <log date="2013-04-29" time="16:57:51">Statistics Generation Starting</log>
        <log date="2013-04-29" time="16:57:51">Built the assets used for collection and calculation</log>
        <log date="2013-04-29" time="16:57:51">Built the phases used in collection and calculation</log>
        <log date="2013-04-29" time="16:57:56">Built the data structure used in collection and calculation
of risk</log>
        <log date="2013-04-29" time="16:58:06">Collected data from output file</log>
        <log date="2013-04-29" time="16:58:06">Completed Asset Statistics Generation</log>
        <log date="2013-04-29" time="16:58:06">Completed Scenario Statistics Generation</log>
        <log date="2013-04-29" time="16:58:06">Completed Capability Statistics Generation</log>
        <log date="2013-04-29" time="16:58:11">Completed Risk Spreadsheet Update</log>
        <log date="2013-04-29" time="16:58:11">Statistics Generation Completed</log>
        <log date="2013-04-29" time="16:58:12">Run Finished</log>
</execution>
```

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Terry Restoule
136 Lewis Street, Suite 1
Ottawa, ON
K2P 0S7

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC APRIL 2011

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Notes on the Conversion of the Tyche Simulation Engine from Visual Basic 6.0 to Visual C#.NET: Tyche 3.0 Development Project

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Restoule, T.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
September 2013	28	1

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Contract Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence Research and Development Canada – CORA
101 Colonel By Drive
Ottawa, Ontario K1A 0K2

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
	W7714-3810

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
N/A	DRDC CORA CR 2013-156

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The objective of this work was to redevelop the Tyche Simulation Engine software using the integrated development environment (Visual C#.NET) recommended in a preceding technical evaluation. Data structures were transformed to reflect the performance results found during the technical evaluation. The software was directly translated to allow the new system to be easily tested in parallel with the original Visual Basic code. Once the new software produced output consistent with the Visual Basic version, the new code was then restructured to improve maintainability. The redeveloped code ran approximately twice as fast as the Visual Basic version during testing.

---

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Tyche; Monte Carlo; Discrete Event Simulation; Visual C#; Performance;

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE DÉFENSE

www.drdc-rddc.gc.ca