aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

2020 Subject & Assessment Guide

Introduction to C#

ICT50215

Diploma of Digital and Interactive Games

academy of interactive entertainment
Sydney Melbourne Canberra Adelaide Lafayette Seattle Online

Table of Contents

Introduction to C#

Units of Competency

The units of competency that are covered in this subject are as follows:

ICTPRG430 – Apply introductory object-oriented language skills

Assessment processes and competency evidence requirements are described in the *Assessment Criteria* section below. If you have prior or other evidence against competency you should discuss this with your teacher.

Subject Overview

Overall Learning Outcomes

- Demonstrate an understanding of the syntax and operation of the C# programming language
- Implement data types, control structures and functions applicable to real-time applications
- Demonstrate a solid understanding of memory management pitfalls and safe coding practices

Subject Description

C#, pronounced C-Sharp, is a managed language that allows a programmer to focus on problem solving rather than on the manipulation low-level resources. Developed in 2000 by Microsoft, C# is one of the most popular programming languages used in software development and is increasingly used in video game programming thanks to popular game engines like MonoGame and Unity 3D.

C# is a general-purpose programming language supporting both the object-oriented and component-oriented programming paradigms. The language makes heavy use of classes and is very similar to C++, another language that has seen widespread adoption within the games industry.

In this subject you will gain a solid grounding in foundational programming concepts common across multiple programming languages. You will learn the syntax and structure of the C# language, and gain practical experience by programming console-based games. This subject also introduces the Unity 3D engine, allowing you to explore the possibilities in 3D game programming enabled by achieving fluency in the C# language.

Industry Relevance

Over the past two decades C# language has seen widespread adoption within the games industry due to its use in popular game engines like Microsoft's XNA, MonoGame, and Unity 3D. C# is also one of the most popular programming languages within the broader software development industries due to its open-source licensing, cross platform support, and integration with may platforms including iOS and Android.

This subject delivers training in foundational programming concepts that are transferrable to a wide variety of modern, object-oriented programming languages found within the game development industry. Skills learnt in this subject are applicable not only to game programmers developing applications and games in C#, but to all programmers regardless of implementation language or platform.

Assumed Knowledge

- A basic understanding of programming concepts and/or some exposure to at least one programming language (not necessarily C++)
- Knowledge of computer use
- Basic logic and problem-solving abilities

Subject Textbooks

The following is a selection of textbooks we recommend for this subject:

- Miles, R, 2016, ***C# Programming Yellow Book***, http://www.csharpcourse.com/
- Nakov, S, Kolev, V, 2013, ***Fundamentals of Computer Programming with C#***, https://introprogramming.info/
- GoalKicker.com, ***C# Notes for Professionals***, https://books.goalkicker.com/CSharpBook/

Assessment Criteria

Assessment Description

Assessment Milestones

Please refer to your Class Schedule for actual dates on your campus

General Description

For this assessment you are to create a text-based program in C# that demonstrates foundational data structures and file handling.

Application

You are to create a text-based console application that meets the specified application brief. The application brief is provided in *Appendix 1*.

To gain competency in this subject, your assessment submission must demonstrate the following components, implemented in the C# programming language:

- Use of variables, operators and expressions
- Use of sequester, selection and iteration
- Functions
- At least two instances of the use of arrays to store primitive or custom data types
- Reading and writing to a text file
- Two classes that each contain four instance variables
- Multiple options for object construction
- User-defined aggregation
- Use of polymorphism once for code extensibility
- Code documentation

By following the provided brief closely you will gather all evidence to demonstrate competency in this subject.

Unit Testing and Debugging

Providing evidence that demonstrates your ability to debug and test your programs is a requirement to gain competency in this subject.

Throughout this subject you will be performing unit testing to confirm each subsystem within your application is functioning correctly. Unit testing should be performed continually throughout the development of your assessment application, and more details can be found in the included project brief in *Appendix 1*.

Evidence Specifications

This is the specific evidence you must prepare for and present by your assessment milestone to demonstrate you have competency in the above knowledge and skills. The evidence must conform to all the specific requirements listed in the table below. You may present additional, or other evidence of competency, but this should be as a result of individual negotiation with your teacher.

Your Roles and Responsibilities as a Candidate

- Understand and feel comfortable with the assessment process.
- Know what evidence you must provide to demonstrate competency.
- Take an active part in the assessment process.
- Collect all competency evidence for presentation when required.

This table defines what you need to produce as evidence of competency.

Assessment Tasks & Evidence Descriptions
1. Application Design Evidence that includes: • A completed application that functions according to the application brief

2. Implementation

Evidence that includes:

- A completed application developed to meet the supplied brief, implemented in C#, demonstrating:
  - A modular approach to implementing logic
  - Class inheritance
  - Two arrays of user defined data types
  - At least two classes containing four or more instance variables
  - At least one instance of an overloaded constructor
  - A class implementing user-defined object aggregation
  - Polymorphism
  - Reading from and writing to a text file
- Use of debugging tools within an IDE to debug and resolve project errors
- Clear code, conforming to a published coding standard
- Well written internal documentation, written according to a published coding standard
- All source code, solution and project files submitted

3. Application Testing

Evidence that includes:

- Project files for at least two stand-alone sub-systems
- Unit test cases for each sub-system submitted, with clear internal documentation

4. Application Handover

Evidence that includes:

- A Visual Studio solution and project that compiles without errors
  - All temporary and built executable files in the obj and bin folder have been removed
- A "readme" or client document explaining how to compile, run and operate the program
- All submitted material archived in a single compressed file (zip, rar, or 7z)

Assessment Instructions for Candidate

METHOD OF ASSESSMENT

Assessment is a cumulative process which takes place throughout a subject. A 'competent' or 'not yet competent' decision is generally made at the end of a subject. Your assessment will be conducted by an official AIE qualified assessor. This may be someone other than your teacher. The evidence you must prepare and present is described

above in this assessment criteria document. This evidence has been mapped to the units of competency listed at the beginning of this document. Assessments will be conducted on a specific milestone recorded above in this assessment guide document.

ASSESSMENT CONDITIONS

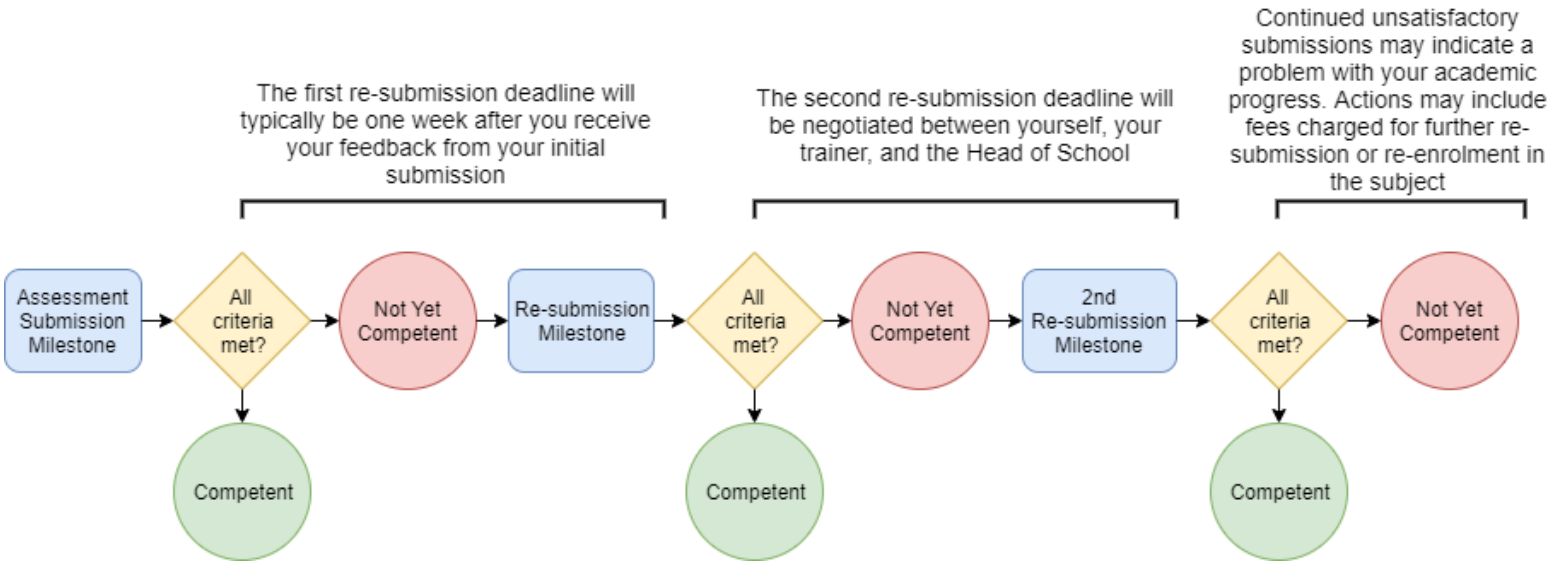
Formative assessment takes place as your teacher observes the development of your work throughout the subject and, although the assessor is likely to be aware of the evidence you are submitting, it is your responsibility to be prepared for the interview where a competency judgement is made (summative assessment). Forgetting something, or making a small mistake at the time of the milestone assessment, can be corrected. However, the assessor may choose to assess other candidates who are better prepared and return to you if time permits.

Upon completion of the assessment you will be issued with feedback and a record of the summative assessment and acknowledge that you have received the result. If you are absent for the nominated assessment milestone (without prior agreement or a sufficiently documented reason) you will be assessed as not yet competent.

GRADING

The assessment you are undertaking will be graded as either *competent* or *not yet competent*.

REASSESSMENT PROCESS



If you are assessed as being not yet competent you will receive clear, written and oral feedback on what you will need to do to achieve competence. Failing to submit an assessment will result in you being assessed as not yet competent. You will be given a reassessment milestone no more than one (1) week later to prepare your evidence. If you are unsuccessful after your reassessment, you may be asked to attend a meeting with your Head of School to discuss your progress or any support you may need and further opportunities to gain competency.

REASONABLE ADJUSTMENTS

We recognise the need to make reasonable adjustments within our assessment and learning environments to meet your individual needs. If you need to speak confidentially to someone about your individual needs, please contact your teacher.

FURTHER INFORMATION

For further information about assessment and support at AIE, please refer to the assessment and course progress sections of your student handbook.

Software

Core

Microsoft Visual Studio

Microsoft's Visual Studio is the recommended IDE for this subject. Other IDEs may be employed if desired as the content of this subject is designed to be cross-platform and IDE agnostic, however we cannot guarantee that all subject material will operate as intended on other IDEs and platforms.

- https://www.visualstudio.com/

Unity3D

Unity3D is a widely used 3D game engine. It has powered many financially and critically successful games. It has a wide array of features that aid with development, especially for a small team. Games made with Unity can be built to a large array of devices. Support for creating custom, in-editor tools is also exposed by the engine.

- http://www.unity3d.com

Appendix 1

Application Brief – RPG Store Simulator

Overview:

Using your knowledge of RPG inventory systems in modern games, create a text-based RPG store simulator using the C# language.

Your store simulator will maintain an array of items (i.e., an inventory) for both the player and the store and allow the player to purchase items from, or sell items to, the store. The inventory arrays of both entities will be updated at the conclusion of each transaction.

Application Description

The store simulation will maintain an inventory of items, from which the player can select. Your player will also have an inventory of items. The player can add to their inventory by purchasing items from the store, or sell an item to the storekeeper.

The player will interact with the storekeeper via a series of text commands. For example, the player may type “View Item” to inspect an item, or “Sell Item” to sell an item in their inventory.

You will use the C# string class and associated functions to process all commands within your game.

You will store each item in an array, and move items between the player’s inventory array and the store’s inventory array as items are bought and sold.

Each item will have a description that is printed to the console when the command to view that item is invoked.

In your game you need to demonstrate multiple levels of inheritance. Create a Base Item class for which all other items types will inherit from. Items of different subclasses should display different information about the item. You can also allow the user to view all items of a specific type. For example, “View Weapons” could display all weapons.

See the cut-down class UML diagram in *figure 1* below that demonstrate this.

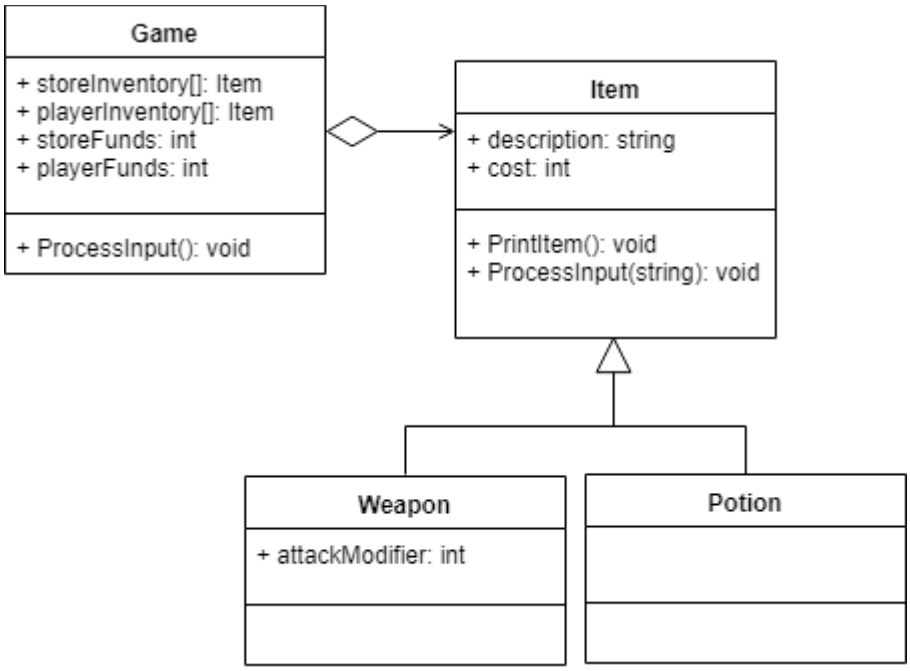


Figure 1. Implement your items as a class hierarchy similar to this.

As a final requirement, your program must save both the player and store inventory to a text file before your program exits, and read the player and store inventories from these text files upon program launch.

Throughout this project you will be performing unit testing to confirm each subsystem is correct before adding it to your simulation.

Requirements:

Your store simulator must have the following features:

- Items are implemented as a custom class, derived from a common base class (see *figure 1*).
- Load the store and player inventory items from a text file when your program starts.
  You may use separate text files for each list.
- Save the store and player inventory items to a text file before the program exits.
  You may use separate text files for each list.
- Use text commands to list the inventory; buy, sell or inspect items.
- Include a hidden command to allow a 'super user' to add new items to the store.
  Implement this by providing an overridden constructor for each item.
- Perform and document unit tests on multiple sub-systems.
  As you implement your text commands, or file I/O, test the system independently before integrating it into your simulation. Ensure you document the unit tests.

- You must write clean, well formatted and well commented source code and provide a "readme" or user document that explains how to compile (for example, which CPU configuration), execute and operate your program.

Extensions:

You should always be looking to push your abilities and expand your knowledge.

The following is a list of optional features you may wish to include in your program to ensure you're maximizing the learning potential of this assessment:

- Display the inventory according to class type. For example, "View Weapons" could display all instances of the Weapon sub-class.
- Sort the inventory according to some condition. You may want to sort by cost, or by attack modifier, for example.
- Recreate your game in Unity 3D. (Note, *only* do this after you have finished the text-based implementation).

Implementation:

With such a large program, it's easy to be overwhelmed and not know where to begin. Below is the suggest strategy for completing this application.

1. Write a program that stores an array of strings. These can be your item names.
   Write the logic that allows a player to search for the name of an item in your array.
   Write and document a unit test to ensure this functionality works correctly.
2. Modify your program to sort your array by alphabetical order.
   Write and document a unit test to ensure this functionality works correctly.
3. Modify your program to add text commands that will allow you to add (sell) and remove (buy) items from the array.
4. Modify your program to read in an array of item names from a text file, then sort the array and write the sorted array back to a new file.
   Write and document a unit test to ensure this functionality works correctly.
5. Add classes to your program. Create the *Item* base class, and a sub-class for each type of inventory item (*Weapon*, *Potion*, *Garment*, *Kittens*, etc).
   Update your text files to include information about the type. For example, a line in your input file might look like:

   Weapon, Spear of Destiny

   Update the inventory array in your program to store *Item* objects, and read in the items from your text file. Write and document a unit test to ensure this functionality works correctly.
6. Add a player inventory, variables for player and store funds, and any additional commands needed to complete your store simulation. Ensure your *Item* class stores at least 4 variables (for example, description, cost, attack modifier, defend modifier, etc).

Submission:

You will need to *submit th*e following:

- A Release build of your application that can execute as a stand-alone program
- Your complete Visual Studio project
- Projects showing unit tests for at least two sub-systems

Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.

Package all files in a single compressed archive file (.zip, .7z, or .rar)

Submission Checklist:

This submission checklist is used to assist your assessor in marking your assessment.

A copy of this checklist can be downloaded from https://aie.instructure.com/. You must complete this checklist yourself and submit it with your project.

General

Description	**Y/N**
All submitted projects compile without errors *Programs that don't compile cannot be assessed*	
The program includes a "readme" or document explaining how to compile, execute and operate the program	
The program performs as described in the general description	
The program contains no logical errors	
The code is sufficiently commented and clean	
An attempt has been made to increase the program's efficiency	
Code compiles without no warnings	
Program executes without crashing	
Program has no memory leaks, and closes all files after use	
A release executable has been made and included in the submission	
Project files and source code are included in the submission	
All files are packaged in a single compressed archive	

Estimate the number of hours taken to complete this assessment	
How many times have you submitted this assessment (including this time)?	

Required Features

Complete the following table by providing the class name or file name, along with the line number, to show where you have implemented each feature.

Feature	**Class/File**	**Line Number**
The program implements a base *Item* class, and two or more sub-classes		
The program stores items in a *player* inventory array and a *store* inventory array		

The program loads both inventory arrays from a text file upon launch		
The program saves both inventory arrays to a text file before exit		
The program contains at least 2 classes containing four or more variables		
The program overrides an object constructor at least once		
The program implements text commands to buy and sell items, and view item descriptions		
Unit testing has been conducted on at least two sub-systems. These projects are included in the submission		
Code is well commented (i.e., each function and class is commented)		

Feature	**Y/N**
Unit testing has been conducted on at least two sub-systems. These projects are included in the submission	
Code is well commented (i.e., each function and class is commented)	

