SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

2021 Subject & Assessment Guide

Introduction to C#

ICT50120 Diploma of Information Technology (Game Programming)

CUA51015 Diploma of Screen and Media

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 complete a series of text-based console applications, as specified in the exercises listed in *Appendix 1* to *Appendix 5.*

Completion of these exercises will demonstrate competence in foundational data structures and file handling, array processing, and debugging.

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. Introductory Exercises Evidence that includes: • Completion of the Introductory Exercises listed in Appendix 1, or other sufficient evidence demonstrating: ○ The application of basic language syntax rules and best practices ○ Selection and use of language data types, operators, and expressions ○ Appropriate language syntax for sequence, selection and iteration constructs ○ Use of a modular programming approach within member or function logic ○ The adherence to organisation guidelines and coding standards for developing maintainable code ○ The application of internal documentation to all the code created
2. Exercises on Arrays Evidence that includes: • Completion of the Exercises on Arrays listed in Appendix 2, or other sufficient evidence demonstrating: ○ The application of arrays, including arrays of objects to introductory programming tasks ○ The use of standard-array processing algorithms ○ The creation and execution of simple tests, to confirm that the code meets the design specification ○ The documenting of tests performed and the results achieved ○ Review and clarification of user requirements with user ○ The ability to plan and determine application design specifications to satisfy user requirements ○ The development of an application according to application design and organisational code conventions

- The adherence to organisation guidelines and coding standards for developing maintainable code
- The application of internal documentation to all the code created

3. Exercises on Files

Evidence that includes:

- Completion of the Exercises on Files listed in Appendix 3, or
  other sufficient evidence demonstrating:
  - Reading and writing data, from and to, text files, and recoding the outcomes
  - The adherence to organisation guidelines and coding standards for developing maintainable code
  - The application of internal documentation to all the code created
  - The creation and execution of simple tests, to confirm that the code meets the design specification
  - The documenting of tests performed and the results achieved

4. Exercises on Classes

Evidence that includes:

- Completion of the Exercises on Classes listed in Appendix 4, or
  other sufficient evidence demonstrating:
  - Implementation of a class that contains primitive member or instance variables
  - Implementation of a class that contains multiple options for object construction
  - Implementation of a class that uses user-defined aggregation
  - Implementation of inheritance, to at least two levels
  - The use of polymorphism at a simple level through inheritance, to enable the easy extension of the code
  - The adherence to organisation guidelines and coding standards for developing maintainable code
  - The application of internal documentation to all the code created
  - The development of a solution, when provided with a basic object-oriented design document

5. Debugging Exercises

Evidence that includes:

- Completion of the Debugging Exercises listed in Appendix 5, or
  other sufficient evidence demonstrating:
  - The use of the debugging facilities of an IDE
  - Interpretation of compiler messages to resolve syntax errors, and the use of debugging techniques to resolve logic errors
  - Referring to the appropriate documentation for the language
  - The use of one debugging tool

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

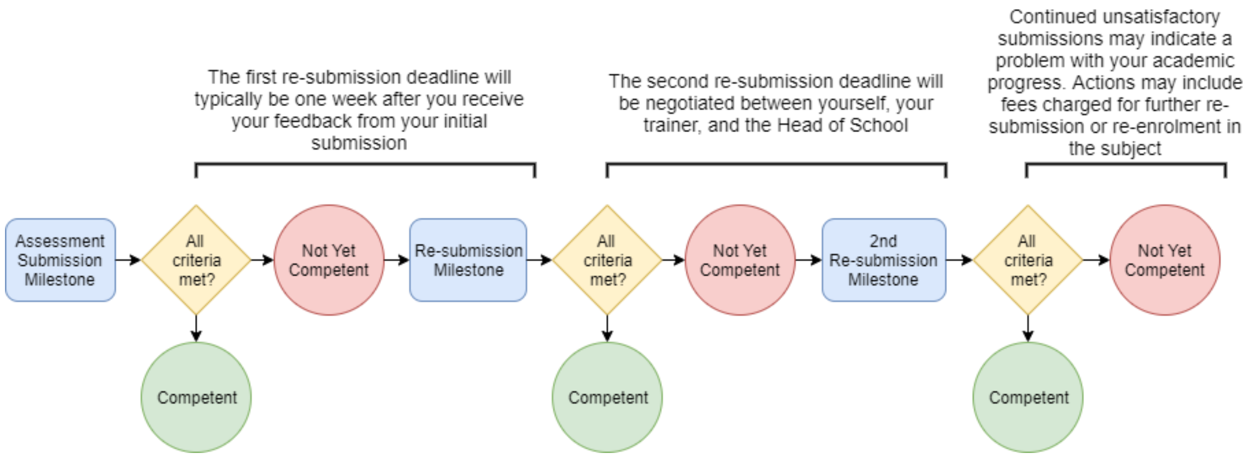
Formative assessment takes place as your teacher observes the development of your work throughout the subject and, although the assessor is likely to be aware of the evidence you are submitting, it is your responsibility to be prepared for the interview where a competency judgement is made (summative 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


The first re-submission deadline will typically be one week after you receive your feedback from your initial submission
The second re-submission deadline will be negotiated between yourself, your trainer, and the Head of School
Continued unsatisfactory submissions may indicate a problem with your academic progress. Actions may include fees charged for further re-submission or re-enrolment in the subject
Assessment Submission Milestone
All criteria met?
Not Yet Competent
Re-submission Milestone
All criteria met?
Not Yet Competent
2nd Re-submission Milestone
All criteria met?
Not Yet Competent
Competent
Competent
Competent

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/

7zip

7-Zip is a free and open-source file archiver, a utility used to place groups of files within compressed containers known as "archives". This utility program will be necessary to package your assessment files for submission.

- https://www.7-zip.org/download.html

Appendix 1

Introductory Exercises

Overview:

Complete the following introductory C# programming exercises.

Exercise 1: Adding Two Numbers

Determine the integer floor of the sum of two floating point numbers. The floor is the truncated float value, i.e. anything after the decimal point is dropped.

Create a function called *AddNumbers* that accepts two parameters of type *float*, and returns an *integer*.

Example

floor(1.1 + 3.89) = floor(4.99) = 4

Exercise 2: Fibonacci

The Fibonacci numbers are a sequence of numbers where each number after the first two is a sum of the prior two. As an illustration, here is a short sequence given starting values of (0, 1):

Fibonacci series = (0, 1, 1, 2, 3, 5, 8, 13)

Given an integer *n*, calculate the first *n* numbers in the Fibonacci sequence given starting elements of *(0, 1)*. Print out *n* integers, including the given *(0, 1)* in the sequence.

Exercise 3: FizzBuzz

Given a number *n*, for each integer *i* in the range from 1 to *n* inclusive, print one value per line as follows:

- If *i* is a multiple of both 3 and 5, print FizzBuzz.
- If *i* is a multiple of 3 (but not 5), print Fizz.
- If *i* is a multiple of 5 (but not 3), print Buzz.
- If *i* is not a multiple of 3 or 5, print the value of *i*.

The function must print the appropriate response for each value i in the set {1, 2, ... n} in ascending order, each on a separate line.

Requirements:

Create a stand-alone console application for each exercise. Where required, have the user input any required information at run time.

Create a testing document to record your testing process. As you test your program, write down the input you pass into the program, the expected output, the actual output, and whether or not the test passed.

Record at least 2 tests for each exercise.

Submission (Manual):

You will need to *submit the* following:

- A Release build of each application that can execute as a stand-alone program
- Your complete Visual Studio project

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 (CodeGrade):

The *CodeGrade* submission tool can be used for these exercises.

If submitting using *CodeGrade*, follow the instructions within each task to ensure your work can be automatically compiled and tested.

You will need to manually submit your testing document.

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**
General	
All submitted projects compile without errors *Programs that don't compile cannot be assessed* **or** All tests passed in *CodeGrade*	
The program includes a "readme" or document explaining how to compile, execute and operate the program, **or** Code submitted using *CodeGrade*	
Your program for each exercise performs as described in the exercise 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, or Code submitted using *CodeGrade*	
A testing document has been created that documents the tests you performed on your program to verify its functionality	

Appendix 2

Exercises on Arrays

Overview:

Complete the following C# programming exercises on arrays.

Exercise 1: Sum of Array

Implement and test a function that accepts an integer array and returns the sum of the array.

```
// Returns the sum of all numbers in the array.
//  - numbers is the array of integers
//  - size is the number of elements in the array

static int SumNumbers(int[] numbers);
```

Elements of *numbers*	Return Value
0, 1, 2, 3	6
2, 4, 8, 16, 32, 64	126
33, 74, 52, 9	168

Exercise 2: Sorting an Array (Descending)

Implement and test a function that sorts the elements in descending order.

```
// Sort the input array in descending order

static void SortDescending(int[] numbers);
```

Before Sorting	After Sorting
4, 3, 2, 1, 0	4, 3, 2, 1, 0
4, 3, 4, 3, 4	4, 4, 4, 3, 3
1, 8, 9, 4, 0, 6, 7, 5	9, 8, 7, 6, 5, 4, 1, 0

Requirements:

Create a stand-alone console application for each exercise. Where required, have the user input any required information at run time.

Create a testing document to record your testing process. As you test your program, write down the input you pass into the program, the expected output, the actual output, and whether or not the test passed.

Record at least 2 tests for each exercise.

Submission (Manual):

You will need to *submit the* following:

- A Release build of each application that can execute as a stand-alone program
- Your complete Visual Studio project

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 (CodeGrade):

The *CodeGrade* submission tool can be used for these exercises.

If submitting using *CodeGrade*, follow the instructions within each task to ensure your work can be automatically compiled and tested.

You will need to manually submit your testing document.

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**
General	
All submitted projects compile without errors *Programs that don't compile cannot be assessed* **or** All tests passed in *CodeGrade*	
The program includes a "readme" or document explaining how to compile, execute and operate the program, **or** Code submitted using *CodeGrade*	
Your program for each exercise performs as described in the exercise 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, or Code submitted using *CodeGrade*	
A testing document has been created that documents the tests you performed on your program to verify its functionality	

Appendix 3

Exercises on Files

Overview:

Complete the following C# programming exercises on files.

Exercise 1: Alphabetize a File

Create a program that will read a plain text file called '*words.txt*' containing a list of randomized words. Your program must sort these words in alphabetical order, and write the ordered list of words to a new file called '*output.txt*'.

The structure of the input file *words.txt* is as follows:

```
13
spillway
fishmonger
bankbook
eyesight
wheelhouse
keyword
pinup
uppercut
washbowl
deadend
coffeemaker
houseboat
typewriter
```

The first line is an integer indicating the number of words in the file. The remaining lines contain one word per line.

The output file *output.txt* that would be produced given this input file is:

```
bankbook
coffeemaker
deadend
eyesight
fishmonger
houseboat
keyword
pinup
spillway
typewriter
uppercut
washbowl
wheelhouse
```

Requirements:

Create a stand-alone console application for this exercise.

Create a testing document to record your testing process. As you test your program, write down the input you pass into the program, the expected output, the actual output, and whether or not the test passed.

You may wish to use the following site to generate a list of random words:
https://www.randomlists.com/random-words

Record at least 2 tests for each exercise.

Submission (Manual):

You will need to *submit the* following:

- A Release build of each application that can execute as a stand-alone program
- Your complete Visual Studio project

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 (CodeGrade):

The *CodeGrade* submission tool can be used for these exercises.

If submitting using *CodeGrade*, follow the instructions within each task to ensure your work can be automatically compiled and tested.

You will need to manually submit your testing document.

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**
General	
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	
Your program for the exercise performs as described in the exercise 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	
A testing document has been created that documents the tests you performed on your program to verify its functionality	

Appendix 4

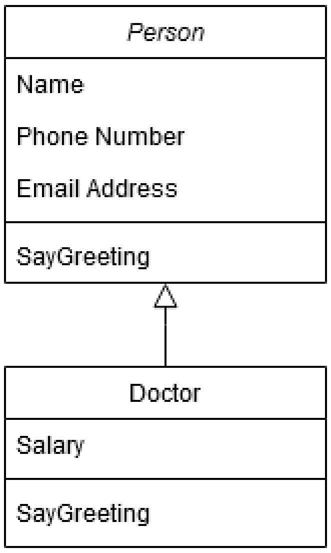
Exercises on Classes

Overview:

Complete the following C# programming exercises on Classes.

Exercise 1: Greetings

Create a *Person* class, and a subclass called *Doctor*.



Each class will have a *SayGreeting* function that will print either the message *"Hello, I'm"* or *"Hello, I'm Dr."* followed by the person's name.

Create a program that initializes an instance of both *Person* and *Doctor*, and outputs their greeting to the console.

For both the *Person* and *Doctor* class, implement both a public default constructor, and a constructor that accepts a string argument containing the person's name.

All classes must be part of the *Greetings* **namespace**.

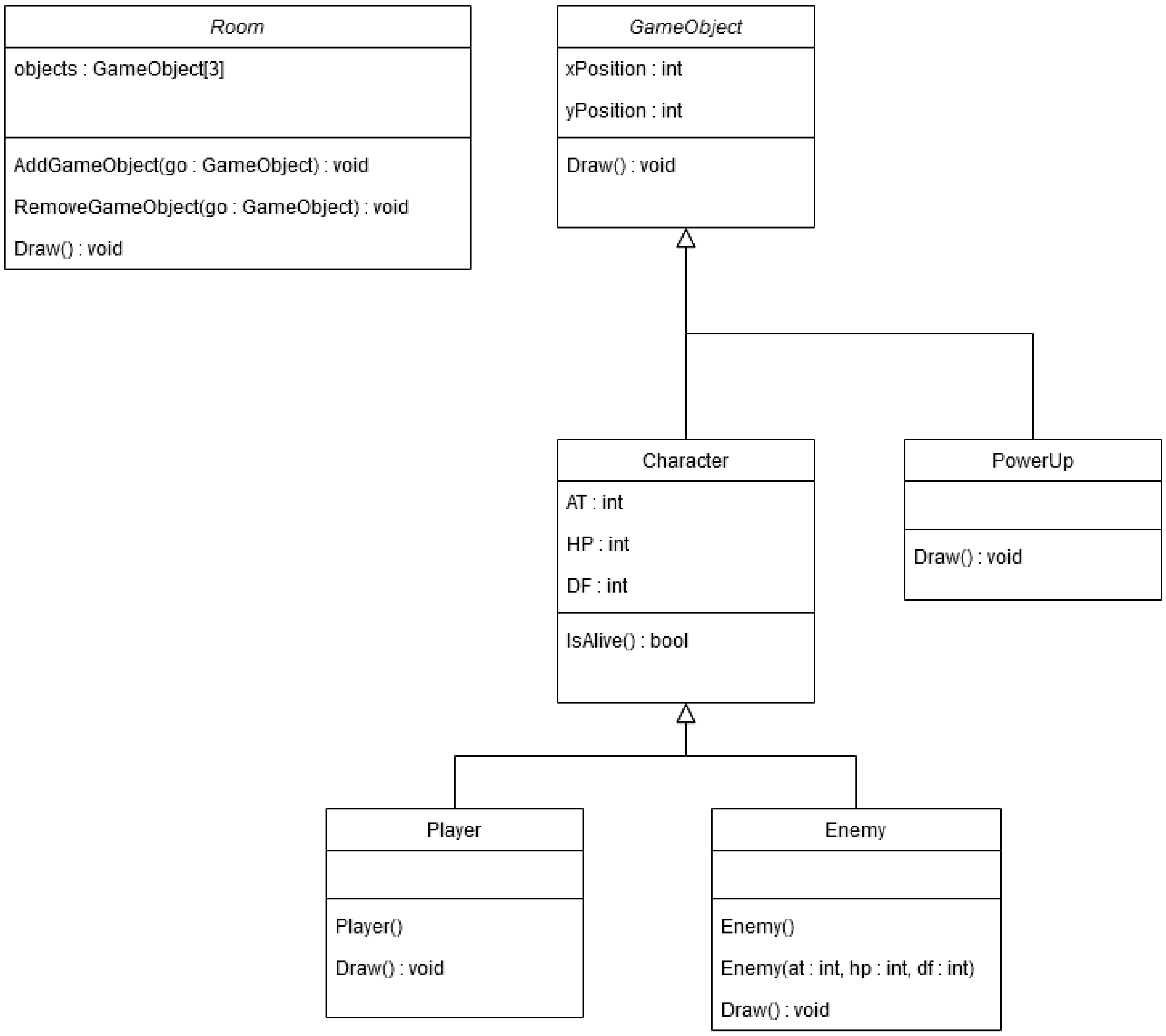
You may add extra functionality to your program if you choose, but you must implement the classes and functions specified in the above class diagram, being careful to use the same namespace, class and function names, and capitalization.

Exercise 2: Adventure Game

For this exercise you will create part of a text-based adventure game.

This game will contain a 2D array of rooms (the map), and each room will either be empty, contain the Player, an Enemy, a Powerup, or any combination of these three.

To create this program, you will need to implement the following classes as described in this class diagram:



Specifications:

Both the *GameObject* and *Character* classes are **abstract classes**.

The *GameObject.Draw()* function is also **abstract**.

All functions are **public**, and all variables should be **protected**.

The *objects* array in the *Room* class is always initialized as an array of 3 *GameObjects*. When the *Room.AddGameObject()* function is called, the input argument is inserted into the first index in the array containing a *null* value.

When objects are removed from the *object* array using *Room.RemoveGameObject()*, the array must be reordered such that any *null* values are at the end of the array. For example, if the value at the first index in the array is removed, all following values must be shifted so that the *null* value occurs at index 2.

When the *Draw()* function is called, it will output a single character to the console (using *Console.Write()*). The character written will depend on the class, and in some cases the class's state.

This is defined as follows:

- *Player.Draw* – writes the character 'X'
- *Enemy*.Draw – writes the character 'O' if HP > 10, otherwise writes 'o'
- *PowerUp.Draw* – writes the character '?'
- *Room*.*Draw* – if empty, writes the character '_'. Otherwise calls *objects[0].Draw()*

All classes must be part of the *AdventureGame* **namespace**.

You may add extra functionality to your program if you choose, but you must implement the classes and functions specified in the above class diagram, being careful to use the same namespace, class and function names, and capitalization.

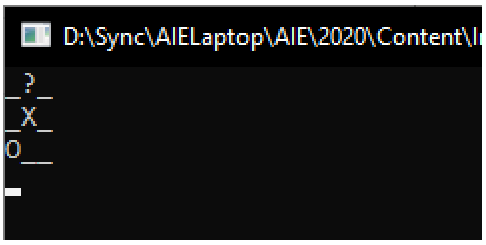
When you have implemented the functionality described above, the following program will display the output given below:

```
namespace AdventureGame {
    class Program {
        static void Main(string[] args) {
            Room[,] map = new Room[3,3];
            for (int row = 0; row < 3; row++) {
                for (int col = 0; col < 3; col++) {
                    map[row, col] = new Room();
                }
            }

            Player p = new Player();
            Enemy e = new Enemy(10, 27, 6);
            PowerUp pu = new PowerUp();

            map[1, 1].AddGameObject(p);
            map[2, 0].AddGameObject(e);
            map[0, 1].AddGameObject(pu);

            for (int row = 0; row < 3; row++) {
                for (int col = 0; col < 3; col++) {
                    map[row, col].Draw();
                }
                Console.WriteLine();
            }
            Console.ReadLine();
        }
    }
}
```


D:\Sync\AIELaptop\AIE\2020\Content\I

```
_?_
_X_
O__
```

Requirements:

Create a stand-alone console application for each exercise.

Submission (Manual):

You will need to *submit the* following:

- A Release build of each application that can execute as a stand-alone program
- Your complete Visual Studio project

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 (CodeGrade):

The *CodeGrade* submission tool can be used for these exercises.

If submitting using *CodeGrade*, follow the instructions within each task to ensure your work can be automatically compiled and tested.

You will need to manually submit your testing document.

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**
General	
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	
Your program for each exercise performs as described in the exercise 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	

Appendix 5

Debugging Exercises

Overview:

Complete the following C# programming exercises on debugging.

These tasks can be completed during the completion of any of the other exercises or programs created for this subject.

Exercise 1: Run the Debugger

Upload a screenshot of Visual Studio running in debug mode.

Exercise 2: Compiler Errors

Upload a screenshot of 3 different compiler errors.

Write a brief description of what each error means.

Exercise 3: Reference Material

List at least one website that provides material to aid you in understanding the C++ language.

Exercise 4: Using the Debugger

Upload a screenshot showing

1. A breakpoint
2. A variable watch, and
3. The callstack