**Assignment Brief – BTEC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Programme** | | Level 3 Extended Diploma in Creative Media Production (Games Development) | | | |
| **Unit number(s) and title covered** | | Unit 71: Object-Oriented Design for Computer Games | | | |
| **Assignment number & title** | | Assignment Two: The Elements of OO Design | | | |
| **Student name** | | *Lewis Hawkins* | | | |
| **Assessor** | | James Shaun | **Internal Verifier** | *David Matravers* | |
| **Date issued** | | *17/01/2019* | **Submission deadline** | *07/02/2019 at* ***4.30pm*** | |
| **Assessment Criteria** | **To achieve the criteria, the evidence must show that the student is able to:** | | | | **Assessor confirm met** |
| P2 | Describe object-oriented design elements with some appropriate use of subject terminology | | | | Yes |
| M2 | Explain object-oriented design elements with reference to detailed illustrative examples and with generally correct use of subject terminology | | | | Yes |
| D2 | Critically assess objectoriented design elements with supporting arguments and elucidated examples, consistently using subject terminology correctly | | | | No |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Assessor feedback - 1st submission** | | | | | | | | | |
| *Task No* | *Targeted Criteria* | *Met* | *Comment* | | | | | | |
| 1 | P2 | Yes | Good start Lewis, you have given some accurate explanations of the structures and features of OO Design, and have used frequent examples as well as code snippets to explain these points, which is why you have been awarded the Merit grade.  You have left some sections a little light, and you do not always fully explain the relevance of the example, or how it fits your point. | | | | | | |
| 1 | M2 | Yes |
| 1 | D2 | No |
| **Did the learner meet the original deadline or agreed extension?** | | | | Yes ☐ | | | | | |
| **Assessor signature** | | | H:\My Pictures\Jim Signature - Copy (2).jpg | | | | **Date** | | 21/02/2019 |
| **Resubmission authorised?** | | | | Yes ☐ No ☐ | | | | | |
| **New agreed deadline date for submission** *\* must be within 10 days of receiving original assignment back* | | | |  | | | | | |
| **Lead Internal Verifier signature** | | |  | | | **Date** | |  | |
| **Assessor feedback - Resubmission** | | | | | | | | | |
|  | | | | | | | | | |
| **Assessor signature** (resubmission only) | | |  | | **Date** | | | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scenario** | | | | |
| You work as a freelance games journalist that has been approached by PCPlayer magazine in Glasgow, who have shown an interest in you to become a monthly columnist in their esteemed publication.  Your editor has asked you to contribute to the publication’s technical tutorials section, to encourage more students to get interested and try out aspects of Object Orientated Design. You should make sure that you cover all aspects required for the brief. The level of detail that you go into, with examples for smoother communication of ideas, will be directly linked to how prominently your article will appear on the company’s website as well a show helpful it will be to the reads. | | | | |
| **Tasks and criteria covered** | | | | |
| **Task 1**  You must write an article which comprehensively explains the elements of object-orientated design for games. This article must include details on the relationships between classes, objects and instances, and how they all fit together in a functioning game environment. You should consider using examples or scripts to help explain your points fully.  You should critically assess examples of the content below, considering the benefits of using object orientated techniques, including inheritance of properties and methods. You may decide to use one game, or many in the process of developing examples to assess the benefits of these techniques   * Structures: *classes, objects, instances, components* * Properties: *class/instance properties; inheritance* * Method header / Method body, *Encapsulation* * Polymorphism: *(overloading, overriding)* * Messaging: *communication between objects*   [Task Covers P2, M2, D2] | | | | |
| **Evidence you must produce for this task** | | | | |
| Report  Supporting Images | | | | |
| **Sources of information** | | | | |
| **Sources of information**  **Textbooks**  Baylis P, Freedman A, Procter N et al – BTEC Level 3 National Creative Media Production, Student Book  (Pearson, 2010) ISBN 978-1846906725  Baylis P, Freedman A, Procter N et al – BTEC Level 3 National Creative Media Production, Teaching Resource  Pack (Pearson, 2010) ISBN 978-1846907371  Gold J— Object-Oriented Game Development (Addison-Wesley, 2004) ISBN 978-0321176608  Makar J – Macromedia Flash MX Game Design Demystified (Macromedia, 2002) ISBN 978-0735713987  Miles R and Hamilton K – Learning UML 2.0 (O’Reilly Media Inc, 2006) ISBN 978-0596009823  Overmars M – ‘Learning Object-Oriented Design by Creating Games’ in Potentials  (the journal of the Institute of Electrical and Electronic Engineers), December 2004-January 2005, Volume 23,  Issue 5, pages 11-13 (available from www.cs.uu.nl/research/techreps/repo/CS-2004/2004-057.pdf)  Rollings A and Morris D – Game Architecture and Design: NRG Programming (New Riders, 2003)  ISBN 978- 0735713634  Swamy N and Swamy N – Basic Game Design and Creation for Fun and Learning (Charles River Media, 2006)  ISBN 978-1584504467  **Websites**  www.cs.uu.nl/research/techreps/repo/CS-2004/2004-057.pdf – article on learning object-oriented design by  creating games, by M Overmars, author of Game Maker software (available from www.yoyogames.com/  make)  www.developer.com/design/ – software development resources and articles  www.devmaster.net/articles/oo-game-design/ – game development encyclopaedia  www.gamasutra.com – respected website for all things game development, sister publication to the respected  print magazine Game Developer; excellent game developer resources  www.macromedia.com/devnet/mx/director/articles/oop\_dir\_flash.html – article on designing and  implementing objects  www.tdan.com/special003.htm – special feature on event progress diagrams | | | | |
| **Student checklist** | | | | **Complete?** |
| Proofread work | | | |  |
| Reference / Bibliography (if applicable) | | | |  |
| All pages attached and numbered – including introduction/conclusion/front sheet | | | |  |
| **Authenticity of Evidence Student declaration** | | | | |
| I certify that the evidence submitted for this assignment is my own.  I have clearly referenced any sources used in the work.  I understand that false declaration of authenticity (i.e. plagiarised work) is a form of academic misconduct and the relevant College procedures will be instigated if I am found to be in contravention of these. | | | | |
| **Student signature** |  | **Date of submission** | 07/02/19 | |
| **Re-authentication of Evidence Student declaration (for resubmission only)** | | | | |
| **Student signature** |  | **Date of resubmission** | 29/03/19 | |

NB. Students – the assignment starts on the first page **after** these front sheets, i.e. Page 1.

* For your convenience, page numbers have been inserted into the footer. **Please keep them**.
* You may choose to add a contents table (ToC) in this section.
* Please **do** **not use text boxes** for the main body of your written answers.
* Please make sure that images/screenshots are correctly formatted, laid out and labelled. A table of Figures (ToF) may also be added if you wish.
* Make sure you use Page (or Section) Breaks whenever a new page is required. (Rather than adding large numbers of Return/Paragraph characters.) Ensure that new Section breaks continue with correct orientation and correct page numbers.
* Ensure that you have referenced your work throughout, using references in text and that you also have a reference list and full bibliography at the end of the work according to the current **Harvard Referencing** conventions. **Failure to do so will make your work more difficult to authenticate.**

**Task One:**

**Structures:**

**Classes:** A class is a “blueprint”, which can be used to build up a game object. It consists of a parent class and as many sub/child classes as you like. A parent class sets up the basic properties which all subclasses would use (e.g. name, icon, description). The properties in the parent class are visible in the child class when attached to an object, which means you can have a flying enemy and a swimming enemy using the same class, without using properties and functions they don’t need. This makes the code clean, optimized and readable.

**Objects:** Objects can be instantiated using classes, which is used to spawn in enemies, chests, etc. The object is created using Classes, components and meshes, which are combined to create anything you want. When creating an enemy, you will need a rigged model (mesh), some animations (walk, run, jump, shoot, idle, reload) which can be applied using avatars, components (nav mesh agent, rigidbody, collider) and classes (AIMasterController, CoverLogic, PlayerDetection, StateMaster).

**Instances:** An instance of an object is also known as a clone of a prefab. It can be instantiated using a class (Instantiate(object, spawn.transform.position, spawn.transform.rotation); which spawns it in the game world at the spawns transforms (location).

In other terms an instance is just a copy of an object that has its own identity, meaning changing a property (health) on one, doesn’t affect other instances (unless directed to).

**Components:** A component is a class that can be added to an object. The class can either be pre-made (rigidbody, collider, nav mesh agent), or custom made by the developer (AIMasterController, CoverLogic, PlayerDetection). They’re used to make the object behave in a specific way, to make the game feel alive and realistic.

**Properties:**

**Class/Instance Properties:**

Class: A class property is a property that is owned by the class itself, therefore all instances of the class hold the property with the same value. Once the value is changed on one instance it changes on all. An example of where this is useful is when using a method or property without wanting a reference. This is done by creating a “static” property or method which can be used anywhere without a reference (since its always the same).

Instance: An instance property is a property that can change on individual instances of an object (hair colour, speed, damage, etc.). Its value is specific to the instance and can change dynamically without affecting other instances in the game world. Instance properties are used to create slight variants in the instances, such as whether an enemy can see the player or not.

**Inheritance:** Inheritance is the transfer of properties and methods from a parent class to a child class (subclass). This is useful for creating different enemies since all enemies use a movement speed variable, but only flying enemies need flying properties / methods.

A parent class is created which holds all properties and any essential methods. Then a subclass is creating by replacing the “MonoBehaviour” with the name of the parent class. The subclass inherits all properties and methods from the parent class, however the parent cannot use properties from the subclass.

Below is an example of a parent (blue) and subclass (yellow):

public class Item : ScriptableObject public class EquippableItem : Item

**Methods:**

**Method Header:** A method header is at the beginning of a method, which defines the type (static, abstract), the publicity (public, private), its Libraries, and Whether it’s a MonoBehaviour or not.

A Library is what tells the class what code to utilise. For example, if you want to use Unity’s UI component, then you would have to implement the using UnityEngine.UI Library.

**Method Body:** The method body is where all the code and properties are stored. It holds all the code the class requires to function (not including subclasses). The method body consists mostly of voids which are encapsulated code to be used in other functions.

**Encapsulation:** Storing code within a function is called encapsulation which is useful for making code easier to read and edit. Once a game has been released it’s likely that the developer would want to update it with more content, which is where making code easy to read and edit becomes essential.

Code is often reused which provides problems if it’s simply copy and pasted. Whether it’s finding bugs or adding new functionality it can be a painful and time-consuming task, which is why encapsulation is so useful.

Saving player progress is a function that is likely to be called under several circumstances, so creating a “public void SaveProgress” will allow for the same code to be called in several places without having to copy and paste. It also allows for the developer to edit the function exclusively without having to worry about finding multiple versions of the same code.

Below is an image of a method, blue is the header, and yellow is the body.

using System.Collections.Generic;

using System.Collections;

using UnityEngine;

public class Item : ScriptableObject

{

public string itemName;

public string description;

public enum Type { Weapon, Armour, Material};

public Type type;

public void AddItem(Inventory inventory)

{

itemSlot.AddItem;

}

}

**Polymorphism:**

**What is Polymorphism?** Polymorphism refers to the ability to present the same interface in different forms. It gives the user flexibility and easy reuse of code, by allowing the same method to have different implementations across different classes.

**Overloading:** Overloading is a type of polymorphism that exists in classes that are independent of each other and completely unrelated (not subclass’). This allows for a method of one class to be used by another without needing any references or connections to the origin.

**Overriding:** Overriding is a type of polymorphism that allows for methods to be called on any class while using the keywords “virtual” and “override”. This allows for a method to be utilised by any script, regardless of its relationship to the origin class.

Below is an example of overloading. Blue = method A, yellow = method B.

  static void GetClossestEnemy(string word)  
    {

word = “Currently”;  
        Debug.Log(word + “Overloading…”);  
    }  
    

static void GetClossestEnemy(string secondWord)  
    {  
        secondWord = “Overload”;

Debug.Log(secondWord + “Complete”);  
    }

Below is an example of overriding. Blue = class A, yellow = class B.

public virtual Item GetCopy() public override Item GetCopy()

{ {

return this; return Instantiate(this);

} }

**Messaging:**

**What is messaging?** Messaging is another way of saying “passing around information”. Whether it’s within a class or between 2 different classes, this is an essential part to events. Below is an example of messaging. Blue = class A, yellow = class B.

if (hit.transform.GetComponentInChildren<AIController>() && !usesProjectle)

{

hit.transform.GetComponentInChildren<AIController>().TakeDamage(damage);

GameObject go = Instantiate(DamageTextPrefab, hit.point, Quaternion.LookRotation(hit.normal));

go.GetComponentInChildren<TextMesh>().text = damage.ToString();

}

public void TakeDamage(float damage)

{

currentHealth -= damage;

state = "Chase";

healthBar.fillAmount = currentHealth / startHealth;

}

In class A a Raycast has been sent out and is checking whether what it’s “hit” has an “AIController” class attached to its “hit” transform. If this returns true the “TakeDamage” function on the AIController is called, with the damage property (on class A) being used to change the damage property on the AIController, and then remove health.

Messaging doesn’t have to be cross classes though, it’s simply the passing of information. When creating a calculator for example messaging is essential, because numbers need to be passed between different functions to add, multiply, etc.

Where it is possible to create a game without messaging, the code would be messy, extensive, and unreliable, because code could not be reused with encapsulation, and only one method could be used. In most instances’ calculator would use only one class, so this wouldn’t be such a big issue, but when creating AI 2, 3 or even 4 classes might be used which is where most of the problems lie.