NIIT Unity Developer Test

## Unity version

* 2019.3

## Expected time to complete

* 2 days

## Items to return

* A zipped-up Unity project folder with your changes
* All documentation containing answers, notes and comments made during the test.

# Introduction

Welcome to the NIIT Unity Developer test.

## Background

Our internally developed comedy platformer "Ninj-haha" really has the client impressed but the developer who created it left the company last week before the project was complete. There are various things wrong with it and a new feature that still needs adding. Can you get the game back on track?

When submitting your project, please include any documentation containing answers and notes that you made during the test. It is much easier to understand what someone was thinking, and why they have implemented something in a certain way if a few notes are included.

A local Git repository is included in the project directory. Please keep this updated as you see fit.

We hope you have fun!

## Control Scheme

WASD – Moves the Ninja.

## Current Functionality Overview

The Ninja can move around the environment.

The Ninja can pick up stars from the floor to increase the score, pick up health packs to increase his health and can hit spears in the ground to decrease his health.

When moving close to some interactable objects (e.g. campfire), the objects respond to the presence of the Ninja.

# **1. Analysis**

As the Ninja moves around the environment, we wanted him to trigger interactions with certain objects. To facilitate this, a simple interaction system was created.

1. Using the campfire as an example, please give a brief overview of how this system works? (3 Marks)

**Ans:** The campfire has the “ParticleSystemInteractable” class component attached to it. The class is a child class of “InteractableObject” which is an abstract class. All InteractableObject classes have an object called “action” of type “IInteractableAction” which have empty Start(), Stop() and Update() functions which are made to be called when the player enters, exits trigger collision box and in Unity’s MonoBehaviour’s Update function which is called every frame respectively.   
  
The “ParticleSystemInteractable” class creates a new object of type “ParticleSystemAction” which implements the interface “IInteractableAction” and assigns it to the “action” object which is declared in the base class “InteractableObject”.   
In the Start function, it the body is written to play the particle effects and in the Stop function code is written to Stop the particle effects which will be called when the player enters and exits the trigger collider accordingly as defined in the “InteractableObject” class.

1. The developer decided to use a well-known design pattern to help implement the interaction system.
   1. Can you identify and describe the design pattern used? (2 Marks)

Ans: All of the InteractableObject classes in the project are using the Strategy pattern which utilises the interface segregation principle of the SOLID principles.

* 1. Why do you think this design pattern was chosen? (2 Mark)

Ans: Strategy and interface segregation principle are the most used patterns in games when developing interactables. It is mainly used because it abstracts the behaviour from the player and encapsulates the behaviour to itself of each interactable. For example, the player can have a function that triggers the interaction function of an interactable object. If there are two interactables such as apple and a phone, when the player goes and interacts with them, the apple will tell the player that it needs to be eaten and the phone will tell the player that it needs to be picked up. This will ensure that the player only knows that they have to interact but does not need to know how as those behaviours will be encapsulated in the interactable classes. If this approach is not used, the player will need to have a huge function to check what interactable are they trying to interact with and how to proceed accordingly.

# **2. Compile Error**

We tried to run the game earlier today but there was a compile error. Please fix this error so that our QA department can continue to test the game. (1 Mark)

Ans: The compilation error was in the CameraController.cs class. This was caused because the class was attempting to modify the individual axis of the position. This is not possible and the whole position needs to be modified instead of trying to change one axis.

# **3. Navigation Issues**

During some testing, the QA department is reporting that there are multiple issues with the current navigation system. As a result, the following bug reports have been filed. Can you investigate and fix them please?

|  |
| --- |
| Bug Report (2 Marks) |
| Description |
| The Ninja cannot walk or run across the first path in the centre of the map. |
| Expected Results |
| The whole environment should be navigable. |
| Actual Results |
| The central dirt path cannot be navigated. |
| Reproduction Steps |
| 1. Use the WASD controls to navigate around the environment. 2. Try to walk across the dirt path in the centre of the map. |
| Accompanying Files |
| [BugReports\CantRunOnPath.mp4](file:///D:\Projects\GameJam\DeveloperTest\CodeTestDoc\BugReports\Cantrunonpath.mp4) |

## **Answer:** I found that the class component “NavMeshModifier” had the Boolean “Override Area” ticked and set the Area Type to “Not Walkable” in the Path gameobject. This was causing the dirth path to be not walkable. To fix it, I simply turned off the “Override Area” Boolean in the “NavMeshModifier” class component.

|  |
| --- |
| Bug Report (2 Marks) |
| Description |
| Obstacles in the environment do not block the player from walking. |
| Expected Results |
| The Ninja should not be able to walk through the pond, trees, walls etc. |
| Actual Results |
| The Ninja character can walk straight through objects that should block his way. |
| Reproduction Steps |
| 1. Use the WASD controls to navigate around the environment. 2. Walk towards an object that should block navigation (i.e. the pond, wall, tree). 3. Walk straight through the object. |
| Accompanying Files |
| [BugReports\CanRunThroughWalls.mp4](file:///D:\Projects\GameJam\DeveloperTest\CodeTestDoc\BugReports\CanRunThroughWalls.mp4) |

**Answer:**

The reason the player was able to walk through these objects was because none of those objects had any sort of colliders.   
To fix the issue, I introduced box colliders to the trees, sphere colliders to the big rocks (I did not put colliders to the small rocks as they were causing un-necessary obstruction when moving around), box collider to the pond and the castle walls.   
I have created a separate gameobject for the north, east and west walls, put a box collider on it and extended it to cover all the walls of that direction. I have done so because this will help in reducing performance issues. For the south wall, since there is a door there, I have introduced two wall colliders to block the side walls of the doors

# **4. Health Bar and Character Controller**

The designers asked for a “Hard Mode” to be added where the Ninja starts with half his maximum health. This mode can be toggled using a Boolean value in the game controller. Testers are reporting that this feature is not working and that the Ninjas health is always full when they start playing.

Can you explain why this is happening and fix it? (3 Marks)

**Ans:** This is happening because the value of the variable “health” is being set in Start() and the SetHealth() function of “CharacterController” class.   
  
In the “GameController” class, in Awake, the SetHealth function is called which sets the health to half of the MaxHealth. But then in the of the Start() function of the “CharacterController” class the health is set to the value of the max health.   
According to this [documentation](https://docs.unity3d.com/Manual/ExecutionOrder.html), we can see that Awake is always called before Start. So even if the value of health is being set to the half of the max health, it then later executes Start() function which sets the health back to max.   
This can be easily fixed by moving the SetHealth() function to be called in the Start function of the “GameController” class. As a general rule of thumb, Awake should be used to assign class variables and Start should be used to modify the variables of those which were assigned in awake.

# **5. Collectable System**

The collectable system is used for star pickups and scoring as well as player hazards and health pickups.

We received the following bug reports for our QA team. Can you investigate and fix these issues?

|  |
| --- |
| Bug Report (2 Marks) |
| Description |
| Each star should only be worth 1 point but every time a star is picked up, 2 is added to the score. |
| Expected Results |
| Only 1 point should be added to the score everything a star is picked up. |
| Actual Results |
| 2 points are being added to the score every time a star is picked up. |
| Reproduction Steps |
| 1. Use the WASD controls to navigate around the environment. 2. Run over to a star. 3. Collect the star by colliding with it. |
| Accompanying Files |
| [BugReports/DoubleScoreStars.mp4](file:///D:\Projects\GameJam\DeveloperTest\CodeTestDoc\BugReports\Doublescorecoins.mp4) |

|  |
| --- |
| Bug Report (3 Marks) |
| Description |
| Stars can be picked up multiple times while they are animating away. |
| Expected Results |
| Only 1 point is added for each star. |
| Actual Results |
| Multiple points can be added for each star. |
| Reproduction Steps |
| 1. Use the WASD controls to navigate around the environment. 2. Run over to a star and collect it. 3. While the star is animating away, move out of the area and back in to score additional point. |
| Accompanying Files |
| [BugReports\DoubleCollectStar.mp4](file:///D:\Projects\GameJam\DeveloperTest\CodeTestDoc\BugReports\Doublecollectcoin.mp4) |

**Answer:**   
The first bug is caused because there are two capsule colliders attached to the Player gameobject.   
   
The second bug was occurring because when the player triggers the OnTriggerEnter function in the collectable class, the trigger collider is not disabled and so when the player goes out and comes back, it triggers the OnTriggerEnter function again.

For both of these bug the solution was to disable the box collider inside the OnPickedUp function.

# **6. Refactor**

It has been identified that there is a circular dependency between the “Collectable” classes and the “CollectableController” class.

1. What is a circular dependency and why is this a bad thing? (3 Marks)

**Answer:** Circular dependency is a condition in software architecture where two modules/classes depend on each other to be setup initially. If both of them rely on each other to be set up, it will never be possible for either of them to be setup as none of them can setup without relying on the other.

1. Update the collectable system to use the observer pattern to remove the circular dependencies. (5 Marks)

**Answer:** I have introduced an interface called “ICollectableControl” which is inherited by “CollectableController” class. The interface has an event and a function. When the “CollectableController” class implements the interface, it fills the body of the interface function to invoke the event.   
  
The setup function of “Collectable” now is expecting the parameter of type “ICollectableControl” instead of “CollectableController” and has a variable of type “ICollectableControl” instead of “CollectableController”.

In the setup function, the “Collectable” class then subscribes a function to the event of “ICollectableControl” which when triggered destroys the required object from the scene.

1. Can you explain the following?
   1. Why is this observer pattern solution better than the original? (1 Mark)

**Answer:** Using the observer pattern, it ensures that the CollectableController’s only function is to tell the collectable objects to call the function that needs to happen when it is picked up.

* 1. Why is the observer pattern a suitable solution to this problem? (1 Mark)

**Answer:** It is suitable because the controller should not be responsible for the behaviour of what each of the collectables should do after they have been picked up.

# **7. Inventory System**

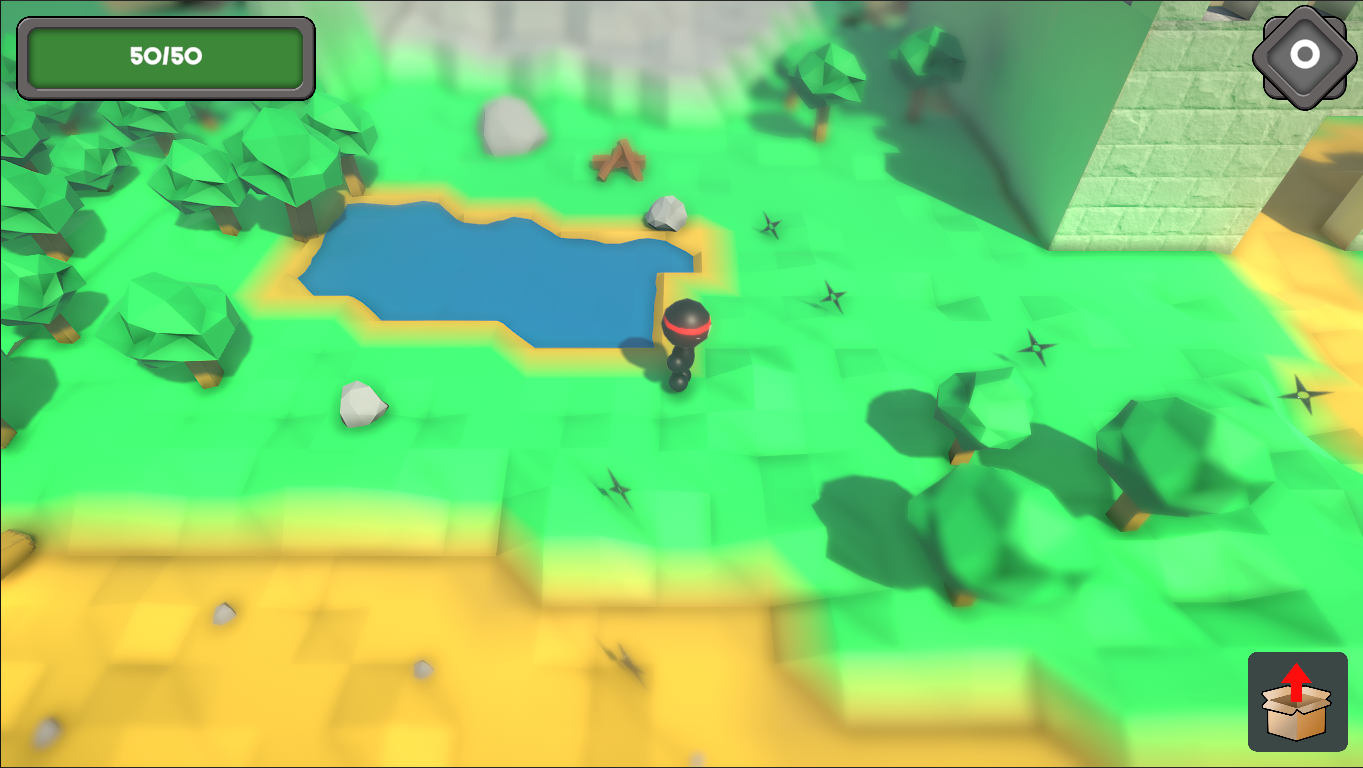
The game is playing a lot better now and the QA department is reporting fewer bugs. Now is a good time to add an inventory system to keep track of the items the player has collected.

1. The section of the Game Design Document below describes how the inventory system should work; can you implement a working version the fits these specifications? (12 Marks)

Bonus marks for using test driven development and including a test suite in your answer (2 Marks)

|  |
| --- |
| Inventory System |
| Overview  The player runs around the environment collecting items that then appear in their inventory.  Requirements   * Needs to keep track of all items collected by the player. * Needs to provide a visual representation of the inventory at any given time.   Features   * Items can be:   + Picked up and added to the inventory.   + Equipped from the inventory into the players hand.   + Dropped back onto the environment. * Items of the same type should only occupy a single slot in the inventory. * The total number of items in the inventory should not exceed 100.   Resources Location   * UI – Assets/InventoryItems/Sprites * Models – Assets/InventoryItems/Fbx |

1. The designers have now completed their final draft of the UI for the new inventory system. Can you follow these designs and implement the inventory system UI into the game please? The UI should remain visually consistent across all screen sizes and aspect ratios so bear this is mind. (6 Marks)





***Answer:* I have created a an inventory system where the player can collect the inventory collectables.The inventory can be accessed by either pressing the “i” button on the keyboard or clicking on the inventory icon on the bottom right of the screen.**

When the player picks up the any of the inventory items, they are collected and can be seen on the inventory screen. If the item being picked up already exists in the inventory, it will just increase the number of instances we have. If a new item is picked up, it will create a new slot.

Clicking on those item slots will make it display more information on the right side of the inventory.  
  
**Conclusion:  
I was not able to work on the Equip and the Drop button functionalities but besides that I have attempted every other challenge.   
  
Test Thoughts:**

**1) Use of prefabs. I had to change every single instance of gameobjects in the scene even though they are literally the same objects.**

**2) Some of the code is unnecessary: Simple use of OOP is enough for most cases.**

**I found Collectable controller to be an extra step that could have worked without it. I see can see the MVC pattern being used here is good for UI but I don't think it really fits well with**

**other use cases.For the UI, it makes sense as the object holding the data and the UI are two different things. But for other cases like the**

**collectables, it seems forced as everything that happens to the collectables are self encapsulated.**

**If there are 50 other types of collectables other than Stars, it would not be fit to create c ontrollers for each of those objects.**

**3) Too many dynamically referenced code usages: Things like getcomponent, add component, find are very expensive and kills the performance. It also allocates**

**unnecessary garbage. In the Character controller, I found the movement variable being created every frame.**

**4) Physics events should be called from inside the FixedUpdate**

**5) I think some places have kind of "forced" the MVC pattern in Unity. For example, the UIController.cs class is using the Find function to find gameobjects to manipulate. This is extremely expensive and could have been made into a monobehaviour where each of these refereneces could be setup in the inspector**

**instead of having to dynamically find them. Instead, in Unity instead of using the traditional C# class constructors we could have just used Monobehaviour and use hard referenced set in the inspector.**