UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Whoosh Division

OcuViz - EpiUse Labs

Unit Testing Plan & Report

Version 1.0

Vukile Langa	u14035449
Wynand Hugo Meiring	u13230795
Nontokozo Hlastwayo	u14414555
Gerome Schutte	u12031519

October 10, 2016

Contents

1 Introduction

1.1 Purpose

OcuViz is a platform for creating rich visual representations of otherwise unintuitive data. It addresses the time consuming task of working with 3D scenes and makes it simple, by providing a format for specifying scenes that is concise, optimised for integration with pluggable data, and is targeted at being easily usable by anyone with even the slightest development experience.

To achieve these goals, the Whoosh Division follows a test-driven development approach for OcuViz. Not only is test write-up an integral part of understanding system objects, but this allows us to ensure that all the parts of the complex system work as intended, notifying us of any breaking changes made while also identifying problematic system components early on in development, enabling us to issue targeted fixes, and giving us confidence in the soundness of system functionality at each release.

This document combines our unit test plan and report into a single coherent artifact.

1.2 Scope

1.3 Test Environment

- **Coding Environment:**
  - **Development Framework:**
    Unity is a development platform used to create interactive 2D, 3D, Virtual- and Augmented Reality experiences, which provides an API to easily create and manipulate scenes and visual objects.
  - **Development Environment:**
    - **Unity Editor:**
      Used to set up and compile the product itself. Behavioural scripts can be imported and attached to objects or invoked at certain events.
    - **Visual Studio Community 2015:**
      The Unity Editor provides the option to integrate a project with Visual Studio. This allows the developer to write scripts and execute compilation commands directly from Visual Studio, while enjoying the benefits Visual Studio provides, such as auto-completion and IntelliSense. Unity Editor also comes packaged with MonoDevelop, a code editor which can be used in the place of Visual Studio.
- **Programming Languages:**
  - Even though Unity supports both **C#** and **JavaScript**, **C#** was chosen as the development language of choice as it provides more readable, intuitive use of object orientation, while supporting optional use of some of the conveniences **JavaScript** provides, such as dynamically typed variables.

- **Testing Frameworks:**
  - **Unity Test Tools:**
    A project asset which provides a framework for Unit and Integration testing. It includes testing annotations, mock object support through the NUnit library, assertion components and integration with the Unity Editor.
- **Other Testing Utilities:**
  - **Editor Test Runner:**
    The Unity Editor categorises both unit- and integration tests as ”editor tests”. This specifies in which directory test scripts need to be stored relative to the project. The Editor Test Runner may be invoked from the Unity Editor, and provides a convenient interface to run tests stored in said location.
  - **Unity Cloud Build:**
    Unity Projects which use GitHub as source control provider may make use of Unity Cloud Build, a continuous integration server specifically for Unity projects. Unity Cloud Build provides the convenience of starting builds as soon as the git repository associated with the project is updated, and runs all editor tests associated with the project on each build. Once the build is completed, results are reported to all collaborators.
- **Operating System:**
  As per architectural requirements set by the client, the build target, development- and testing environment is x64 Windows.

1.4 Assumptions and Dependencies

1. **Assumptions:**
   - **System component packaging:**
     All of the system components to be tested are assumed to belong to the EntityProvider package.
   - **System component access rights:**
     All of the system components to be tested are assumed to have public access rights. This is so that testing suites may be interchangeable without having to include the testing suite in the EntityProvider package itself.
   - **Programming language soundness:**
     It is assumed that all of the functionality and structures provided by the C# language are consistent and work as documented when used as intended.
   - **Utility soundness:**
     It is assumed that all of the functionality and structures provided by the Unity Test Tools asset are consistent and work as documented when used as intended. If a test case fails, it is assumed that it is not due to error on the part of the structures provided in the asset. Asset errors are assumed to be verbose, and results are assumed to be repeatable under the same environment conditions.

- **Environment soundness:**
  It is assumed that all of the functionality and structures provided by the Unity Editor and x64 Windows are consistent and work as documented when used as intended. If a test case fails, it is assumed that it is not due to error on the part of the environment implementation. Environment errors are assumed to be verbose, and results are assumed to be repeatable under the same environment conditions.

2. **Dependencies:**

   - **Unity:**
     The project cannot be compiled without a Unity supporting environment, such as Unity Editor or Unity Cloud build. While it is not necessary to build the entire project every time tests must be run, scripts under inspection must be compiled and require access to specific Unity APIs.
   - **Unity Test Tools:**
     The unit tests used in the project are built around the assertion component and NUnit, provided with Unity Test Tools. As they are required to be present in the project by Unity Cloud Build, this asset is included in the project source control and does not need to be included separately.

Unit Test Plan

2 Test Items

Due to the nature of the project, and being heavily reliant on the Unity game engine, we decided to take the approach of testing the individual classes and their methods to ensure that they behaved as expected. This being said, we attempted to check values of objects created by the game engine as well as we could and as far as Unity would allow.

3 Functional Features to be Tested

4 Test Cases

4.1 Test Case 1: CollectionFactory

4.1.1 Condition 1: build_returnsCollection

Objective: Test whether the class CollectionFactory does indeed return a collection of Entities to be used in a scene.

Input: The following entity was used to create a collection:

1. string[] list: { "", "", "row", "12", "0","0","0"}.

Outcome: The following outcomes are expected to pass the test:

1. It returns a collection ready to make copies of an Entity (design states this Entity should be specified later)
2. It should create 12 copies of the prototype Entity.

4.1.2 Condition 2: build_throwsArgumentNullException

Objective: Test whether the class CollectionFactory does indeed throw an ArgumentNullException if a null parameter is passed into CollectionFactory.build().

Input: The following entity was used to create a collection:

1. string[] list: null.

Outcome: The following outcomes are expected to pass the test:

1. It throws an ArgumentNullException.

4.1.3 Condition 3: build_throwsInvalidListLengthException

Objective: Test whether the class CollectionFactory does indeed throw an InvalidListLength-Exception if a string array with length not equal to 7 is passed into CollectionFactory.build(). This is to ensure that the factory receives an exact number of parameters when creating a collection.

Input: The following entity was used to create a collection:

1. string[] list: { "", "", "row", "12", "0","0","0", null}

Outcome: The following outcomes are expected to pass the test:

1. It throws an InvalidListLengthException.

4.1.4 Condition 4: buildBasic_returnsCollection

Objective: Test whether the class CollectionFactory returns a collection of only one Entity. This method is only to be used by the editor through the front-end user interface.

Input: The following values were sent into Collection.buildBasic():

1. an empty string for button pressed (this value is insignificant to CollectionFactory, but specified by abstract parent EntityFactory)
2. "collection" for the name of the collection
3. "3d", to create a basic 3D collection (the editor should make necessary changes as the user wants).

Outcome: The following outcomes are expected to pass the test:

1. It returns a non-null object
2. The object is of type Entity
3. The object is of type Collection.

4.1.5 Condition 5: buildBasic_throwsArgumentNullException1

Objective: Test whether the class CollectionFactory throws an ArgumentNullException if the second parameter is null.

Input: The following values were sent into Collection.buildBasic():

1. string button: ""

2. string entityLink: null
3. string type: "3d".

Outcome: The following outcomes are expected to pass the test:

1. It throws an ArgumentNullException due to the name of the collection being null.

4.1.6 Condition 6: buildBasic_throwsArgumentNullException1

Objective: Test whether the class CollectionFactory throws an ArgumentNullException if the third parameter is null.

Input: The following values were sent into Collection.buildBasic():

1. string button: ""
2. string entityLink: ""
3. string type: null.

Outcome: The following outcomes are expected to pass the test:

1. It throws an ArgumentNullException due to the type of the collection being null.

4.2 Collection

4.2.1 Condition 1: createCollection_returnsCollection

Objective: Test whether the class Collection returns a collection of Entities in the form of a List when commanded to do so. This is done during runtime by the EntityProvider to only create a collection when rendering the scene.

Input: This method takes no input. All necessary information needed to create the collection is already existing within the Collection instance.

Outcome: The following outcomes are expected to pass the test:

1. It returns a non-null object
2. The object is an instance of List<Entity>.

4.2.2 Condition 2: setEntity_setsEntity

Objective: Tests if Collection.setEntity() sets the prototype Entity as expected. This is done during runtime by the EntityProvider to only create a collection when rendering the scene.

Input:

1. Entity entity, entity.name == "Proto".

Outcome: The following outcomes are expected to pass the test:

1. The Entity set is the same as the one passed into the Collection.

4.2.3 Condition 3: setEntity_throwsArgumentNullException

Objective: Tests if Collection.setEntity() throws an ArgumentNullException if a null Entity is passed. This could cause a NullReferenceException later when attempting to clone the Entity should it be null.

Input:

1. Entity entity: null.

Outcome: The following outcomes are expected to pass the test:

1. The Collection throws an ArgumentNullException.

4.2.4 Condition 4: setType_setsType

Objective: Tests if Collection.setType() sets the type of collection to be created.

Input:

1. string type: "stack".

Outcome: The following outcomes are expected to pass the test:

1. The Collection type matches the type passed into it.

4.2.5 Condition 5: setType_throwsCollectionTypeNotFoundException

Objective: Tests if Collection.setType() throws a CollectionTypeNotFoundException if the type passed is not recognised or expected by the Collection.

Input:

1. string type: "undefined".

Outcome: The following outcomes are expected to pass the test:

1. The Collection throws a CollectionTypeNotFoundException.

4.2.6 Condition 6: setDimension_setsDimension

Objective: Tests if Collection.setDimension() sets the dimension of the collection to an undefined integer. This ensures that no negative values can be passed into the collection.

Input:

- uint dimension: 10.

Outcome: The following outcomes are expected to pass the test:

1. The Collection returns a dimension matching the one input.

4.3 Colour

4.3.1 Condition 1: Colour_createsColour

Objective: This test checks if the constructor of Colour creates a colour object used by Entities based on the parameters passed, using the name and the hexadecimal value of the colour.

Input: There were two colour objects created:

1. string name: "black", string hex: "#iii"
2. string name: "huh", string hex: "#00f".

Outcome: The following outcomes are expected to pass the test:

1. The colour creates a black Unity Color object. This is deduced from the name used
2. The colour corresponding to the hex value #00f, or blue. This is because the name ("huh") is not recognised as a predefined type.

4.3.2 Condition 2: Colour_throwsArgumentNullException1

Objective: This test checks if the constructor of Colour throws an ArgumentNullException if the name of the colour is null.

Input:

1. string name: null
2. string hex: "#iii".

Outcome: The following outcomes are expected to pass the test:

1. The constructor throws an ArgumentNullException.

4.3.3 Condition 3: Colour_throwsArgumentNullException2

Objective: This test checks if the constructor of Colour throws an ArgumentNullException if the name of the hexadecimal value is null.

Input:

1. string name: ""
2. string hex: null.

Outcome: The following outcomes are expected to pass the test:

1. The constructor throws an ArgumentNullException.

4.3.4 Condition 4: Colour_throwsInvalidColourHexException

Objective: This test checks if the constructor of Colour throws an InvalidColourHexException if the name of the hexadecimal value and the colour are unknown.

Input:

1. string name: ""
2. string hex: null.

Outcome: The following outcomes are expected to pass the test:

1. The constructor throws an InvalidColourHexException.

4.4 CommaTokeniser

4.4.1 Condition 1: tokenise_tokenises

Objective: This test checks if the string CommaTokeniser returns an array of tokenised strings, using ',' as a separator.

Input:

1. string line: "Hello,there".

Outcome: The following outcomes are expected to pass the test:

1. CommaTokeniser.tokenise() returns an instance of a string array
2. The length of the array is more than one
3. The length of the array is equal to two.

4.4.2 Condition 2: tokenise_throwsNullReferenceException

Objective: This test checks if CommaTokeniser throws a NullReferenceException if it receives a null value to tokenise.

Input:

1. string line: null.

Outcome: The following outcomes are expected to pass the test:

1. CommaTokeniser.tokenise() throws a NullReferenceException.

4.4.3 Condition 3: tokenise_throwsListSeparatorNotFoundException

Objective: This test checks if CommaTokeniser throws a ListSeparatorNotFoundException if it receives a string with no comma.

Input:

1. string line: "a bunch of gibberish with no comma?".

Outcome: The following outcomes are expected to pass the test:

1. CommaTokeniser.tokenise() throws a ListSeparatorNotFoundException.

4.5 ConcreteEntityPool

4.5.1 Condition 1: indexOf_NotFound_ReturnsInvalidIndex

Objective: A unit test which checks whether ConcreteEntityPool.indexOf returns a not-found index when calling it with an argument which indicates an entity which is not present in the EntityPool.

Input:

1. string entityName: ”notfound”.

Outcome: The following outcomes are expected to pass the test:

1. ConcreteEntityPool.indexOf returns -1 if no entity named ”notfound” is in the Entity pool.

4.5.2 Condition 2: indexOf_Found_ReturnsValidIndex

Objective: A unit test which checks whether ConcreteEntityPool.indexOf returns a valid index when calling it with an argument which indicates an entity which is present in the EntityPool.

Input:

1. string entityName: ”found”.

Outcome: The following outcomes are expected to pass the test:

1. ConcreteEntityPool.indexOf returns 0 if the Entity is found at index 0.

4.5.3 Condition 3: store_NoDuplicates_StoresEntity

Objective: A unit test which checks whether ConcreteEntityPool.store correctly adds an entity to the pool when calling it with an argument entity which is not present in the EntityPool.

Input:

1. Entity entity: Entity foo, with Entity.name == ”added”.

Outcome: The following outcomes are expected to pass the test:

1. Adds the Entity if no duplicates are present.

4.5.4 Condition 4: store_Duplicates_ThrowsException

Objective: A unit test which checks whether ConcreteEntityPool.store correctly raises an exception when calling it with an argument entity which is present in the EntityPool.

Input: The following entities were attempted to be added into the ConcreteEntity-Pool:

1. Entity entity: Entity foo, with Entity.name == ”added”
2. Entity entity: Entity bar, with Entity.name == ”added”.

Outcome: The following outcome is expected to pass the test:

1. ConcreteEntityPool throws DuplicateEntityException.

4.5.5 Condition 5: fetch_EntityExists_FetchesCorrectReference

Objective: A unit test which checks whether ConcreteEntityPool.fetch correctly fetches an entity reference when calling it with an argument entity name which indicates an entity that is present in the EntityPool.

Input: The following entity was added into the ConcreteEntityPool:

1. Entity entity: Entity expected, with Entity.name == ”added”.

Outcome: The following outcome is expected to pass the test:

1. ConcreteEntityPool returned Entity expected.

4.5.6 Condition 6: fetch_EntityNotExist_ThrowsException

Objective: A unit test which checks whether ConcreteEntityPool.fetch correctly raises an exception when calling it with an argument entity name which indicates an entity that is not present in the EntityPool.

Input: There was no input sent into the pool.

Outcome: The following outcome is expected to pass the test:

1. ConcreteEntityPool throws an EntityNotFoundException.

4.6 CustomCollection

4.6.1 Condition 1: setOriginal_setsOriginalEntity

Objective: This test ensures that the Custom Collection Factory sets the entity to be used as a default when creating the collection.

Input: The following values were passed into CollectionFactory.setOriginal():

1. Entity entity: Entity.name == ”BoO! It’s !Halloween”
2. Tokeniser token: new CommaTokeniser()
3. Reader reader: new Reader().

Outcome: The following outcome is expected to pass the test:

1. The Entity returned matches the entity passed into the method.

4.6.2 Condition 2: setOriginal_throwsArgumentNullException1

Objective: This test ensures that the Custom Collection Factory throws an ArgumentNullException if the entity being set is null.

Input: The following values were passed into CollectionFactory.setOriginal():

1. Entity entity: null
2. Tokeniser token: new CommaTokeniser()
3. Reader reader: new Reader().

Outcome: The following outcome is expected to pass the test:

1. The factory throws an ArgumentNullException.

4.6.3 Condition 3: setOriginal_throwsArgumentNullException2

Objective: This test ensures that the Custom Collection Factory throws an ArgumentNullException if the tokeniser being passed is null is null.

Input: The following values were passed into CollectionFactory.setOriginal():

- Entity entity: Entity entity: Entity.name == ”BoO! It’s !Halloween”
- Tokeniser token: null
- Reader reader: new Reader().

Outcome: The following outcome is expected to pass the test:

1. The factory throws an ArgumentNullException.

4.6.4 Condition 4: setOriginal_throwsArgumentNullException1

Objective: This test ensures that the Custom Collection Factory throws an ArgumentNullException if the FileReader being passed is null.

Input: The following values were passed into CollectionFactory.setOriginal():

1. Entity entity: Entity entity: Entity.name == ”BoO! It’s !Halloween. Not really :/”
2. Tokeniser token: new CommaTokeniser()
3. Reader reader: null.

Outcome: The following outcome is expected to pass the test:

1. The factory throws an ArgumentNullException.

4.6.5 Condition 5: build_returnsInstanceOfEntity

Objective: This test ensures that the method CustomCollectionFactory.build() returns an instance of Entity, and instance of CustomCollection.

Input: The following string array was passed into CollectionFactory.build():

1. list[0]: ”0”
2. list[1]: ”name”
3. list[2]: ”C:Assets\\CSV\\Scene2Input1.csv”.

Outcome: The following outcomes are expected to pass the test:

1. The Entity returned by setOriginal() matches the entity passed into the method
2. The object returned by build() is an Entity
3. The object returned by build() is a CustomCollection.

4.6.6 Condition 6: build_throwsInvalidListLengthException

Objective: This test ensures that CustomCollectionFactory.build() throws InvalidListLength-Exception.

Input: A string array of length 4 was passed, when one of length 3 was expected.

Outcome: The following outcomes are expected to pass the test:

1. An InvalidListLengthException must be thrown.

4.6.7 Condition 7: build_throwsArgumentNullException

Objective: This test ensures that CustomCollectionFactory.build() throws Argument-NullException.

Input: A null string array was passed. **Outcome:** The following outcomes are expected to pass the test:

1. An ArgumentNullException must be thrown.

4.7 EntityProvider

4.7.1 Condition 1: createGameObject_createsBasicStackCollection

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic Collection of type stack.

Input: The following values were used for testing:

1. string button: ”shapes”
2. string entityLink: ”test1”
3. string type: ”stack”

Outcome: The following outcomes are expected to pass the test:

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”stack”.

4.7.2 Condition 2: createGameObject_createsBasicRandomCollection

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic Collection of type random.

Input: The following values were used for testing:

1. string button: ”shapes”
2. string entityLink: ”test1”
3. string type: ”random”

Outcome: The following outcomes are expected to pass the test:

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”random”.

4.7.3 Condition 3: createGameObject_createsBasicRowCollection

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic Collection of type row.

Input: The following values were used for testing:

1. string button: ”shapes”
2. string entityLink: ”test1”
3. string type: ”row”

Outcome: The following outcomes are expected to pass the test:

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”row”.

4.7.4 Condition 4: createGameObject_createsBasic3DCollection

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic 3D Collection.

Input: The following values were used for testing:

1. string button: "shapes"
2. string entityLink: "test1"
3. string type: "3d"

Outcome: The following outcomes are expected to pass the test:

1. The object returned is an instance of Collection
2. The type of Collection.getType() is "3d".

4.7.5 Condition 5: createGameObject_createsBasic2DCollection

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic 2D Collection.

Input: The following values were used for testing:

1. string button: "shapes"
2. string entityLink: "test1"
3. string type: "2d"

Outcome: The following outcomes are expected to pass the test:

1. The object returned is an instance of Collection
2. The type of Collection.getType() is "2d".

4.7.6 Condition 6: createGameObject_failsToCreateBasicModel

Objective: This test ensures that EntityProvider.CreateGameObject() throws a NotImplementedException when attempting to create a basic model, as this is not allowed. Models may only be imported.

Input: The following values were used for testing:

1. string button: "2d"
2. string entityLink: "test1"

3. string type: ”model”

Outcome: The following outcomes are expected to pass the test:

1. NotImplementedException must be thrown.

4.7.7 Condition 7: createGameObject_returnsBasicShape1

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic shape Entity.

Input: The following values were used for testing:

1. string button: ”3d”
2. string entityLink: ”test1”
3. string type: ”plane”

Outcome: The following outcomes are expected to pass the test:

1. An instance of Entity is returned
2. The Entity’s GameObject has the name passed into EntityProvider.CreateGameObject().

4.7.8 Condition 8: createGameObject_returnsBasicShape2

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic shape Entity.

Input: The following values were used for testing:

1. string button: ”3d”
2. string entityLink: ”test1”
3. string type: ”cube”

Outcome: The following outcomes are expected to pass the test:

1. An instance of Entity is returned
2. The Entity’s GameObject has the name passed into EntityProvider.CreateGameObject().

4.7.9 Condition 9: createGameObject_returnsBasicShape3

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic shape Entity.

Input: The following values were used for testing:

1. string button: ”3d”
2. string entityLink: ”test1”
3. string type: ”sphere”

Outcome: The following outcomes are expected to pass the test:

1. An instance of Entity is returned
2. The Entity’s GameObject has the name passed into EntityProvider.CreateGameObject().

4.7.10 Condition 10: createGameObject_returnsBasicShape4

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic shape Entity.

Input: The following values were used for testing:

1. string button: ”3d”
2. string entityLink: ”test1”
3. string type: ”capsule”

Outcome: The following outcomes are expected to pass the test:

1. An instance of Entity is returned
2. The Entity’s GameObject has the name passed into EntityProvider.CreateGameObject().

4.7.11 Condition 10: createGameObject_returnsBasicShape5

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic shape Entity.

Input: The following values were used for testing:

1. string button: ”3d”
2. string entityLink: ”test1”

3. string type: ”cylinder”

Outcome: The following outcomes are expected to pass the test:

1. An instance of Entity is returned
2. The Entity’s GameObject has the name passed into EntityProvider.CreateGameObject().

4.7.12 Condition 11: createGameObject_returnsBasicShape6

Objective: This test ensures that EntityProvider.CreateGameObject() returns a basic shape Entity.

Input: The following values were used for testing:

1. string button: ”3d”
2. string entityLink: ”test1”
3. string type: ”quad”

Outcome: The following outcomes are expected to pass the test:

1. An instance of Entity is returned
2. The Entity’s GameObject has the name passed into EntityProvider.CreateGameObject().

4.7.13 Condition 12: createGameObject_throwsShapeTypeNotFoundException

Objective: This test ensures that EntityProvider.CreateGameObject() throws ShapeTypeNotFoundException when an invalid type is used.

Input: The following values were used for testing:

1. string button: ”3d”
2. string entityLink: ”test1”
3. string type: ”ellipse”

Outcome: The following outcomes are expected to pass the test:

1. ShapeTypeNotFoundException must be thrown.

4.7.14 Condition 13: createGameObject_createsBasicAreaLight

Objective: This test ensures that EntityProvider.CreateGameObject() returns a Light of type area.

Input: The following values were used for testing:

1. string button: ”area”
2. string entityLink: ”test1”
3. string type: ”quad”

Outcome: The following outcomes are expected to pass the test:

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Area.

4.7.15 Condition 14: createGameObject_createsBasicSpotLight

Objective: This test ensures that EntityProvider.CreateGameObject() returns a Light of type spotlight.

Input: The following values were used for testing:

1. string button: ”spot”
2. string entityLink: ”test1”
3. string type: ”quad”

Outcome: The following outcomes are expected to pass the test:

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Spotlight.

4.7.16 Condition 15: createGameObject_createsBasicDirectionalLight

Objective: This test ensures that EntityProvider.CreateGameObject() returns a Light of type directional.

Input: The following values were used for testing:

1. string button: ”directional”
2. string entityLink: ”test1”

3. string type: ”quad”

Outcome: The following outcomes are expected to pass the test:

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Directional.

4.7.17 Condition 16: createGameObject_createsBasicPointLight

Objective: This test ensures that EntityProvider.CreateGameObject() returns a Light of type point.

Input: The following values were used for testing:

1. string button: ”point”
2. string entityLink: ”test1”
3. string type: ”quad”

Outcome: The following outcomes are expected to pass the test:

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Point.

4.7.18 Condition 17: createGameObject_cannotFindButton

Objective: This test ensures that EntityProvider.CreateGameObject() throws an ArgumentException if it receives an unknown button.

Input: The following values were used for testing:

1. string button: ”soft ice cream”
2. string entityLink: ””
3. string type: ”type”

Outcome: The following outcomes are expected to pass the test:

1. ArgumentException is thrown.

4.7.19 Condition 18: createGameObject_throwsArgumentNullException1

Objective: This test ensures that EntityProvider.CreateGameObject() throws an ArgumentNullException if it receives a null button.

Input: The following values were used for testing:

1. string button: null
2. string entityLink: ””
3. string type: ”type”

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

4.7.20 Condition 19: createGameObject_throwsArgumentNullException2

Objective: This test ensures that EntityProvider.CreateGameObject() throws an ArgumentNullException if it receives a null name.

Input: The following values were used for testing:

1. string button: ””
2. string entityLink: null
3. string type: ”type”

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

4.7.21 Condition 20: createGameObject_throwsArgumentNullException3

Objective: This test ensures that EntityProvider.CreateGameObject() throws an ArgumentNullException if it receives a null type.

Input: The following values were used for testing:

1. string button: ””
2. string entityLink: ””
3. string type: null

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

4.7.22 Condition 21: setBackgroundColour_setsColourToSkybox

Objective: This test ensures that EntityProvider.SetBackground() sets the background colour of the scene to a sky box, complete with a sun disc and ocean.

Input: The following values were used for testing:

1. string colour: "skybox".

Outcome: The following outcomes are expected to pass the test:

1. The canvas' camera.clearFlags equals CameraClearFlags.Skybox
2. The viewers' camera.clearFlags equals CameraClearFlags.Skybox.

4.7.23 Condition 22: setBackgroundColour_setsColourToSolidColor

Objective: This test ensures that EntityProvider.SetBackground() sets the background colour of the scene to a solid color.

Input: The following values were used for testing:

1. string colour: "#000".

Outcome: The following outcomes are expected to pass the test:

1. The canvas' camera.clearFlags equals CameraClearFlags.SolidColor
2. The viewers' camera.clearFlags equals CameraClearFlags.SolidColor.

4.7.24 Condition 23: setBackgroundColour_setsColourToHexColour

Objective: This test ensures that EntityProvider.SetBackground() sets the background colour of the scene to a specified hexadecimal colour.

Input: The following values were used for testing:

1. string colour: "#000".

Outcome: The following outcomes are expected to pass the test:

1. The canvas' camera.backgroundColor equals #000
2. The viewers' camera.backgroundColor equals #000.

4.7.25 Condition 24: storeEntity_storesEntity

Objective: This test ensures that EntityProvider.storeEntity() stores the given Entity into EntityProvider's pool.

Input: The following values were used for testing:

1. Entity entity: Entity entity, entity.name == ”entity link goes here”.

Outcome: The following outcomes are expected to pass the test:

1. The EntityPool's size was larger than 0
2. The object fetched from the pool is the same stored
3. The object fetched has the name ”entity link goes here”.

4.7.26 Condition 25: storeEntity_throwsArgumentNullException

Objective: This test ensures that EntityProvider.storeEntity() throws an ArgumentNullException if a null object is being attempted to be stored.

Input: The following values were used for testing:

1. Entity entity: null.

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

4.7.27 Condition 26: renderScene_placesPoolIntoScene

Objective: This test ensures that EntityProvider.renderScene() places the Entities in the pool into the scene.

Input: The following values were used for testing:

1. An array of Entities looped to be created an placed in the pool via EntityProvider.storeEntity().

Outcome: The following outcomes are expected to pass the test:

1. Each Entity's GameObject stored that was to be stored in EntityProvider.entityPool is checked against all the GameObjects in the scene. All must match.

4.7.28 Condition 27: renderScene_cannotPlaceNullObjects1

Objective: This test ensures that EntityProvider.renderScene() fails to place a null object in the scene.

Input: The following values were used for testing:

1. An array of Entities looped to be created an placed in the pool via EntityProvider.storeEntity()
2. The 8th object is set to null.

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

4.8 Entity

4.8.1 Condition 1: setGameObject_setsGameObject

Objective: Objective is to ensure Entity.setGameObject() sets the GameObject within the Entity to the GameObject required.

Input: The following values were used for testing:

1. GameObject obj: GameObject go, go.name == "Pokemon".

Outcome: The following outcomes are expected to pass the test:

1. Entity.GetGameObject() returns GameObject equal to go.

4.8.2 Condition 2: setGameObject_throwsNullReferenceException

Objective: Objective is to ensure Entity.setGameObject() throws a NullReferenceException when the GameObject is null.

Input: The following values were used for testing:

1. GameObject obj: null.

Outcome: The following outcomes are expected to pass the test:

1. NullReferenceException is thrown.

4.8.3 Condition 3: setName_setsName

Objective: Objective is to ensure Entity.setName() sets the name of the Entity to the string passed.

Input: The following values were used for testing:

1. string name: "Pokemon".

Outcome: The following outcomes are expected to pass the test:

1. Entity.getName() returns "Pokemon".

4.8.4 Condition 4: setName_throwsNullReferenceException

Objective: Objective is to ensure Entity.setName() throws a NullReferenceException when the name is null.

Input: The following values were used for testing:

1. string name: null.

Outcome: The following outcomes are expected to pass the test:

1. NullReferenceException is thrown.

4.8.5 Condition 5: addColour_addsColour

Objective: Objective is to ensure Entity.addColour() sets the colour of the GameObject within the Entity to the colour required.

Input: The following values were used for testing:

1. Colour obj: Colour colour, colour.name == "blue", colour.hex == "#00f".

Outcome: The following outcomes are expected to pass the test:

1. The GameObject's renderer's material color is blue.

4.8.6 Condition 6: addColour_throwsNullReferenceException

Objective: Objective is to ensure Entity.setGameObject() throws a NullReferenceException when the Colour is null.

Input: The following values were used for testing:

1. Colour obj: null.

Outcome: The following outcomes are expected to pass the test:

1. NullReferenceException is thrown.

4.8.7 Condition 7: addTexture_findsTexture

Objective: Objective is to ensure Entity.addTexture() sets the texture of the Entity to the texture path passed, if found.

Input: The following values were used for testing:

1. string path: "moon_surface".

Outcome: The following outcomes are expected to pass the test:

1. No exceptions are thrown by the system or the Unity game engine.

4.8.8 Condition 8: addTexture_throwsNullReferenceException

Objective: Objective is to ensure Entity.addTexture() throws a NullReferenceException when the texture path is null.

Input: The following values were used for testing:

1. string path: null.

Outcome: The following outcomes are expected to pass the test:

1. NullReferenceException is thrown.

4.9 FactoryShop

4.9.1 Condition 1: getFactory_returnsCollectionFactory

Objective: Ensure FactoryShop.getFactory() returns a CollectionFactory when requested.

Input: The following values were used for testing:

1. string typeName: "Collection".

Outcome: The following outcomes are expected to pass the test:

1. Returns an instance of EntityFactory
2. Returns a CollectionFactory.

4.9.2 Condition 2: getFactory_returnsCustomCollectionFactory

Objective: Ensure FactoryShop.getFactory() returns a CustomCollectionFactory when requested.

Input: The following values were used for testing:

1. string typeName: "CustomCollection".

Outcome: The following outcomes are expected to pass the test:

1. Returns an instance of EntityFactory
2. Returns a CustomCollectionFactory.

4.9.3 Condition 3: getFactory_returnsModelFactory

Objective: Ensure FactoryShop.getFactory() returns a ModelFactory when requested.

Input: The following values were used for testing:

1. string typeName: "Model".

Outcome: The following outcomes are expected to pass the test:

1. Returns an instance of EntityFactory
2. Returns a ModelFactory.

4.9.4 Condition 4: getFactory_returnsLightFactory

Objective: Ensure FactoryShop.getFactory() returns a LightFactory when requested.

Input: The following values were used for testing:

1. string typeName: "Light".

Outcome: The following outcomes are expected to pass the test:

1. Returns an instance of EntityFactory
2. Returns a LightFactory.

4.9.5 Condition 5: getFactory_returnsShapeFactory

Objective: Ensure FactoryShop.getFactory() returns a ShapeFactory when requested.

Input: The following values were used for testing:

1. string typeName: ”Shape”.

Outcome: The following outcomes are expected to pass the test:

1. Returns an instance of EntityFactory
2. Returns a ShapeFactory.

4.9.6 Condition 6: getFactory_returnsViewerFactory

Objective: Ensure FactoryShop.getFactory() returns a ViewerFactory when requested.

Input: The following values were used for testing:

1. string typeName: ”Viewer”.

Outcome: The following outcomes are expected to pass the test:

1. Returns an instance of EntityFactory
2. Returns a ViewerFactory.

4.9.7 Condition 7: getFactory_throwsArgumentNullException

Objective: Ensure FactoryShop.getFactory() throws ArgumentNullException if type is null.

Input: The following values were used for testing:

1. string typeName: null.

Outcome: The following outcomes are expected to pass the test:

1. Throws ArgumentNullException.

4.9.8 Condition 8: getFactory_throwsArgumentException

Objective: Ensure FactoryShop.getFactory() throws ArgumentException if type is not known.

Input: The following values were used for testing:

1. string typeName: "undefined factory".

Outcome: The following outcomes are expected to pass the test:

1. Throws ArgumentException.

4.10 FileReader

4.10.1 Condition 1: getLines_returnsLinesOfScene1

Objective: Ensure FileReader.getLines() returns a List of lines from the first scene's CSV file.

Input: The following values were used for testing:

1. string fileName: "C:Assets\\CSV\\Scene1.csv".

Outcome: The following outcomes are expected to pass the test:

1. Returns a List of strings, containing the lines of the CSV file.

4.10.2 Condition 2: getLines_throwsArgumentNullException

Objective: Ensure FileReader.getLines() throws ArgumentNullException if the file name is null.

Input: The following values were used for testing:

1. string fileName: null.

Outcome: The following outcomes are expected to pass the test:

1. Throws ArgumentNullException.

4.10.3 Condition 3: getLines_throwsFileNotFoundException

Objective: Ensure FileReader.getLines() throws FileNotFoundException if the file is not found.

Input: The following values were used for testing:

1. string fileName: "C:Assets\\CSV\\Scene1.csv".

Outcome: The following outcomes are expected to pass the test:

1. Throws FileNotFoundException.

4.11 LightFactory

4.11.1 Condition 1: build_buildsSpotlight

Objective: This test is to verify LightFactory.build() returns a spotlight when one is requested.

Input: The following values were used for testing:

1. string list[10]: {"", "not", "", "spot", "#000edd", "0", "0", "0", "0", "0"}.

Outcome: The following outcomes are expected to pass the test:

1. The light component of the Entity's GameObject returned is of type Spot.

4.11.2 Condition 2: build_buildsAreaLight

Objective: This test is to verify LightFactory.build() returns an area light when one is requested.

Input: The following values were used for testing:

1. string list[10]: {"", "not", "", "area", "#000edd", "0", "0", "0", "0", "0"}.

Outcome: The following outcomes are expected to pass the test:

1. The light component of the Entity's GameObject returned is of type Area.

4.11.3 Condition 3: build_buildsDirectionalLight

Objective: This test is to verify LightFactory.build() returns a directional light when one is requested.

Input: The following values were used for testing:

1. string list[10]: {"", "not", "", "directional", "#000edd", "0", "0", "0", "0", "0"}.

Outcome: The following outcomes are expected to pass the test:

1. The light component of the Entity's GameObject returned is of type Directional.

4.11.4 Condition 4: build_buildsPoint

Objective: This test is to verify LightFactory.build() returns a point light when one is requested.

Input: The following values were used for testing:

1. string list[10]: {"", "not", "", "point", "#000edd", "0", "0", "0", "0", "0"}.

Outcome: The following outcomes are expected to pass the test:

1. The light component of the Entity's GameObject returned is of type Point.

4.11.5 Condition 5: build_throwsLightTypeNotFoundException

Objective: This test is to verify LightFactory.build() throws LightTypeNotFoundException if a light type that is unknown is requested.

Input: The following values were used for testing:

1. string list[10]: {"", "not", "", "ray", "#000edd", "0", "0", "0", "0", "0"}.

Outcome: The following outcomes are expected to pass the test:

1. Throws LightTypeNotFoundException.

4.11.6 Condition 6: build_throwsInvalidListLengthException

Objective: This test is to verify LightFactory.build() throws InvalidListLengthException if a the list of parameters is not the expected length.

Input: The following values were used for testing:

1. string list[11]: {"", "not", "", "ray", "#000edd", "0", "0", "0", "0", "0", null}.

Outcome: The following outcomes are expected to pass the test:

1. Throws InvalidListLengthException.

4.11.7 Condition 7: build_throwsArgumentNullException

Objective: This test is to verify LightFactory.build() throws ArgumentNullException if a the list of parameters is null.

Input: The following values were used for testing:

1. string list[11]: null.

Outcome: The following outcomes are expected to pass the test:

1. Throws ArgumentNullException.

4.11.8 Condition 8: buildBasic_buildsSpotlight

Objective: This test is to verify LightFactory.buildBasic() returns a basic spotlight with defaults.

Input: The following values were used for testing:

1. string button: ”spot”
2. string entityLink: ”liht”
3. string type: ”spot”.

Outcome: The following outcomes are expected to pass the test:

1. Returns a basic Entity with a light component of type Spot.

4.11.9 Condition 9: buildBasic_buildsAreaLight

Objective: This test is to verify LightFactory.buildBasic() returns a basic area light with defaults.

Input: The following values were used for testing:

1. string button: ”area”
2. string entityLink: ”liht”
3. string type: ”area”.

Outcome: The following outcomes are expected to pass the test:

1. Returns a basic Entity with a light component of type Area.

4.11.10 Condition 10: buildBasic_buildsDirectionalLight

Objective: This test is to verify LightFactory.buildBasic() returns a basic directional light with defaults.

Input: The following values were used for testing:

1. string button: ”directional”
2. string entityLink: ”liht”
3. string type: ”directional”.

Outcome: The following outcomes are expected to pass the test:

1. Returns a basic Entity with a light component of type Directional.

4.11.11 Condition 11: buildBasic_buildsPoint

Objective: This test is to verify LightFactory.buildBasic() returns a basic point light with defaults.

Input: The following values were used for testing:

1. string button: "point"
2. string entityLink: "liht"
3. string type: "point".

Outcome: The following outcomes are expected to pass the test:

1. Returns a basic Entity with a light component of type Point.

4.11.12 Condition 12: buildBasic_throwsLightTypeNotFoundException

Objective: This test is to verify LightFactory.buildBasic() throws LightTypeNotFoundException if an unrecognised light type is requested.

Input: The following values were used for testing:

1. string button: "ray"
2. string entityLink: "liht"
3. string type: "ray".

Outcome: The following outcomes are expected to pass the test:

1. Throws LightTypeNotFoundException.

4.11.13 Condition 13: buildBasic_throwsArgumentNullException1

Objective: This test is to verify LightFactory.buildBasic() throws ArgumentNullException if the entityLink value is null.

Input: The following values were used for testing:

1. string button: ""
2. string entityLink: null
3. string type: "ray".

Outcome: The following outcomes are expected to pass the test:

1. Throws ArgumentNullException.

4.11.14 Condition 14: buildBasic_throwsArgumentNullException2

Objective: This test is to verify LightFactory.buildBasic() throws ArgumentNullException if the type value is null.

Input: The following values were used for testing:

1. string button: null
2. string entityLink: ”liht”
3. string type: null.

Outcome: The following outcomes are expected to pass the test:

1. Throws ArgumentNullException.

4.12 ModelFactory

4.12.1 Condition 1: build_returnsEntity

Objective: This test is to verify ModelFactory.build() returns an Entity containing the model, should the model be found.

Input: The following values were used for testing:

1. string list[9]:{””, ”model”, ”C:Assets\\Resources\\david.obj”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1”}.

Outcome: The following outcomes are expected to pass the test:

1. Returns an Entity instance.

4.12.2 Condition 2: build_throwsInvalidListLengthException

Objective: This test is to verify ModelFactory.build() throws an InvalidListLengthException if a list not of length 9 is provided.

Input: The following values were used for testing:

1. string list[8]:{””, ”model”, ”C:Assets\\Resources\\david.obj”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1”, ”1”}.

Outcome: The following outcomes are expected to pass the test:

1. Throws InvalidListLengthException.

4.12.3 Condition 3: build_throwsArgumentNullException

Objective: This test is to verify ModelFactory.build() throws an ArgumentNullException if a list is null.

Input: The following values were used for testing:

1. string list[9]: null.

Outcome: The following outcomes are expected to pass the test:

1. Throws ArgumentNullException.

4.12.4 Condition 4: build_throwsDirectoryNotFoundException

Objective: This test is to verify ModelFactory.build() throws an DirectoryNotFoundException if a model's directory is not found.

Input: The following values were used for testing:

1. string list[9]:{"", "model", "C:Assets\\SO\\notDavid.obj", "1", "1", "1", "1", "1", "1", "1", "1"}.

Outcome: The following outcomes are expected to pass the test:

1. Throws DirectoryNotFoundException.

4.12.5 Condition 5: build_throwsFileNotFoundException

Objective: This test is to verify ModelFactory.build() throws an FileNotFoundException if a model's directory is not found.

Input: The following values were used for testing:

1. string list[9]:{"", "model", "C:Assets\\notDavid.obj", "1", "1", "1", "1", "1", "1", "1", "1"}.

Outcome: The following outcomes are expected to pass the test:

1. Throws FileNotFoundException.

4.13 ShapeFactory

4.13.1 Condition 1: build_returnsEntityContainingGameObject

Objective: This test is to verify ShapeFactory.build() returns an Entity with a GameObject encapsulated.

Input: The following values were used for testing:

1. string list[13]:{"", "name", "null", "plane", "true", "true", "100", "1", "1", "1", "1", "1", "1"}.

Outcome: The following outcomes are expected to pass the test:

1. An Entity is returned
2. The Entity has a non-null GameObject within.

4.13.2 Condition 2: build_throwsArgumentNullException

Objective: This test is to verify ShapeFactory.build() throws an ArgumentNullException if the list is null.

Input: The following values were used for testing:

1. string list[13]: null.

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

4.13.3 Condition 3: build_throwsInvalidListLengthException

Objective: This test is to verify ShapeFactory.build() throws an InvalidListLengthException if the list is not an expected length of 13.

Input: The following values were used for testing:

1. string list[12]: {"", "name", "null", "plane", "true", "true", "100", "1", "1", "1", "1", "1"}.

Outcome: The following outcomes are expected to pass the test:

1. InvalidListLengthException is thrown.

4.14 ViewerFactory

4.14.1 Condition 1: build_returnsEntityConstainingViewer

Objective: This test is to verify ViewerFactory.build() returns an Entity with the viewer GameObject encapsulated, to be used to change the parameters of the viewer dynamically.

Input: The following values were used for testing:

1. string[] list = { "", "viewer", "8", "10", "12", "0", "180", "0" }.

Outcome: The following outcomes are expected to pass the test:

1. The object returned is not null
2. An Entity's GameObject's location is Vector3(8, 10, 12);
3. The Entity has a GameObject named "RigidBodyFPSController".

4.14.2 Condition 2: build_throwsArgumentNullException

Objective: This test is to verify ViewerFactory.build() throws an ArgumentNullException if the list passed is null.

Input: The following values were used for testing:

1. string[] list = null.

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

4.14.3 Condition 3: build_throwsInvalidListLengthException

Objective: This test is to verify ViewerFactory.build() throws an InvalidListLengthException if the list passed has a length other than 8.

Input: The following values were used for testing:

1. string[] list = { "", "viewer", "8", "10", "12", "0", "180", "0", "some unexpected value" }.

Outcome: The following outcomes are expected to pass the test:

1. InvalidListLengthException is thrown.

4.14.4 Condition 4: buildBasic_returnsDefaultViewer

Objective: This test is to verify ViewerFactory.buildBasic() returns an Entity with the viewer GameObject encapsulated, to be used to change the parameters of the viewer dynamically.

Input: The following values were used for testing:

1. string button: ""

2. string entityLink: ”viewer”
3. string type: ””.

Outcome: The following outcomes are expected to pass the test:

1. The object returned is not null
2. An Entity is returned
3. The Entity has a GameObject within, named ”RigidBodyFPSController”.

4.14.5 Condition 5: buildBasic_throwsArgumentNullException

Objective: This test is to verify ViewerFactory.build() throws an ArgumentNullException if the entityLink passed is null.

Input: The following values were used for testing:

1. string button: ””
2. string entityLink: null
3. string type: ””.

Outcome: The following outcomes are expected to pass the test:

1. ArgumentNullException is thrown.

5 Item Pass/Fail Criteria

Each item tested must meet a certain criteria in order to pass. These criteria are as follows:

- The outcomes of the criteria mentioned in each test case should be met
- These are tested using Assert statements in the UnityEngine, as well as testing thrown exceptions against expected exceptions.

If any of the above criteria are not met, the item will be considered failed.

6 Test Deliverables

Artefacts to be produced as part of unit testing include the following:

- Unit Test Plan
- Unit Test Report
- Link to Unit Test Code

Unit Test Report

7 Detailed Test Results

7.1 Overview of Test Results

7.2 Functional Requirements Test Results

7.2.1 Test Case 1 (4.1.1)

The following results were obtained from the tests conducted. The tests to produce the following results have passed/failed and the reasons are stated below.

1. It returns a collection ready to make copies of an Entity (design states this Entity should be specied later)
2. It should create 12 copies of the prototype Entity.

Result: Pass.

7.2.2 Test Case 2 (4.1.2)

1. It throws an ArugmentNullException.

Result: Pass.

7.2.3 Test Case 3 (4.1.3)

1. It throws an InvalidListLengthException.

Result: Pass.

7.2.4 Test Case 4 (4.1.4)

1. It returns a non-null object
2. The object is of type Entity
3. The object is of type Collection.

Result: Pass.

7.2.5 Test Case 5 (4.1.5)

1. It throws an ArgumentNullException due to the name of the collection being null.

Result: Pass.

7.2.6 Test Case 6 (4.1.6)

1. It throws an ArgumentNullException due to the type of the collection being null.

Result: Pass.

7.2.7 Test Case 7 (4.2.1)

1. It throws an ArgumentNullException due to the type of the collection being null.

Result: Pass.

7.2.8 Test Case 8 (4.2.2)

1. The Entity set is the same as the one passed into the Collection.

Result: Pass.

7.2.9 Test Case 9 (4.2.3)

1. The Collection throws an ArgumentNullException.

Result: Pass.

7.2.10 Test Case 10 (4.2.4)

1. The Collection type matches the type passed into it.

Result: Pass.

7.2.11 Test Case 11 (4.2.5)

1. The Collection throws a CollectionTypeNotFoundException.

7.2.12 Test Case 12 (4.2.6)

1. The Collection returns a dimension matching the one input.

Result: Pass.

7.2.13 Test Case 13 (4.3.1)

1. The colour creates a black Unity Color object. This is deduced from the name used
2. The colour corresponding to the hex value #00f, or blue. This is because the name (huh) is not recognised as a predened type.

Result: Pass.

7.2.14 Test Case 14 (4.3.2)

1. The constructor throws an ArgumentNullException.

Result: Pass.

7.2.15 Test Case 15 (4.3.3)

1. The constructor throws an ArgumentNullException.

Result: Pass.

7.2.16 Test Case 16 (4.3.4)

1. The constructor throws an InvalidColourHexException.

Result: Pass.

7.2.17 Test Case 17 (4.4.1)

1. CommaTokeniser.tokenise() returns an instance of a string array
2. The length of the array is more than one
3. The length of the array is equal to two.

Result: Pass.

7.2.18 Test Case 18 (4.4.2)

1. CommaTokeniser.tokenise() throws a NullReferenceException.

Result: Pass.

7.2.19 Test Case 19 (4.4.3)

1. CommaTokeniser.tokenise() throws a ListSeparatorNotFoundException.

Result: Pass.

7.2.20 Test Case 20 (4.5.1)

1. ConcreteEntityPool.indexOf returns -1 if no entity named notfound is in the Entity pool.

Result: Pass.

7.2.21 Test Case 21 (4.5.2)

1. ConcreteEntityPool.indexOf returns 0 if the Entity is found at index 0.

Result: Pass.

7.2.22 Test Case 22 (4.5.3)

1. Adds the Entity if no duplicates are present.

Result: Pass.

7.2.23 Test Case 23 (4.5.4)

1. ConcreteEntityPool throws DuplicateEntityException.

Result: Pass.

7.2.24 Test Case 24 (4.5.5)

1. ConcreteEntityPool returned Entity expected.

Result: Pass.

7.2.25 Test Case 25 (4.5.6)

1. ConcreteEntityPool throws an EntityNotFoundException.

Result: Pass.

7.2.26 Test Case 26 (4.6.1)

1. The Entity returned matches the entity passed into the method.

Result: Pass.

7.2.27 Test Case 27 (4.6.2)

1. The factory throws an ArgumentNullException.

Result: Pass.

7.2.28 Test Case 28 (4.6.3)

1. The factory throws an ArgumentNullException.

Result: Pass.

7.2.29 Test Case 29 (4.6.4)

1. The factory throws an ArgumentNullException.

Result: Pass.

7.2.30 Test Case 30 (4.6.5)

1. The Entity returned by setOriginal() matches the entity passed into the method
2. The object returned by build() is an Entity
3. The object returned by build() is a CustomCollection.

Result: Pass.

7.2.31 Test Case 31 (4.6.6)

1. An InvalidListLengthException must be thrown.

Result: Pass.

7.2.32 Test Case 32 (4.6.7)

1. An ArgumentNullException must be thrown.

Result: Pass.

7.2.33 Test Case 33 (4.7.1)

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”stack”.

Result: Pass.

7.2.34 Test Case 34 (4.7.2)

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”random”.

Result: Pass.

7.2.35 Test Case 35 (4.7.3)

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”row”.

Result: Pass.

7.2.36 Test Case 36 (4.7.4)

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”3d”.

Result: Pass.

7.2.37 Test Case 37 (4.7.5)

1. The object returned is an instance of Collection
2. The type of Collection.getType() is ”2d”.

Result: Pass.

7.2.38 Test Case 38 (4.7.6)

1. NotImplementedException must be thrown.

Result: Pass.

7.2.39 Test Case 39 (4.7.7)

1. An instance of Entity is returned
2. The Entitys GameObject has the name passed into EntityProvider.CreateGameObject().

Result: Pass.

7.2.40 Test Case 40 (4.7.8)

1. An instance of Entity is returned
2. The Entitys GameObject has the name passed into EntityProvider.CreateGameObject().

Result: Pass.

7.2.41 Test Case 41 (4.7.9)

1. An instance of Entity is returned
2. The Entitys GameObject has the name passed into EntityProvider.CreateGameObject().

Result: Pass.

7.2.42 Test Case 42 (4.7.10)

1. An instance of Entity is returned
2. The Entitys GameObject has the name passed into EntityProvider.CreateGameObject().

Result: Pass.

7.2.43 Test Case 43 (4.7.11)

1. An instance of Entity is returned
2. The Entitys GameObject has the name passed into EntityProvider.CreateGameObject().

Result: Pass.

7.2.44 Test Case 44 (4.7.12)

1. An instance of Entity is returned
2. The Entitys GameObject has the name passed into EntityProvider.CreateGameObject().

Result: Pass.

7.2.45 Test Case 45 (4.7.13)

1. ShapeTypeNotFoundException must be thrown.

Result: Pass.

7.2.46 Test Case 46 (4.7.14)

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Area.

Result: Pass.

7.2.47 Test Case 47 (4.7.15)

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Spotlight.

Result: Pass.

7.2.48 Test Case 48 (4.7.16)

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Directional.

Result: Pass.

7.2.49 Test Case 49 (4.7.17)

1. Object must be an instance of Entity
2. The Entity should contain a Light GameObject with LightType Point.

Result: Pass.

7.2.50 Test Case 50 (4.7.18)

1. ArgumentException is thrown.

Result: Pass.

7.2.51 Test Case 51 (4.7.19)

1. ArgumentNullException is thrown.

Result: Pass.

7.2.52 Test Case 52 (4.7.20)

1. ArgumentNullException is thrown.

Result: Pass.

7.2.53 Test Case 53 (4.7.21)

1. ArgumentNullException is thrown.

Result: Pass.

7.2.54 Test Case 54 (4.7.22)

1. The canvas' camera.clearFlags equals CameraClearFlags.Skybox
2. The viewers' camera.clearFlags equals CameraClearFlags.Skybox.

Result: Pass.

7.2.55 Test Case 55 (4.7.23)

1. The canvas' camera.clearFlags equals CameraClearFlags.SolidColor
2. The viewers' camera.clearFlags equals CameraClearFlags.SolidColor.

Result: Pass.

7.2.56 Test Case 56 (4.7.24)

1. The canvas' camera.backgroundColor equals #000
2. The viewers' camera.backgroundColor equals #000.

Result: Pass.

7.2.57 Test Case 57 (4.7.25)

1. The EntityPools size was larger than 0
2. The object fetched from the pool is the same stored
3. The object fetched has the name ”entity link goes here”.

Result: Pass.

7.2.58 Test Case 58 (4.7.26)

1. ArgumentNullException is thrown.

Result: Pass.

7.2.59 Test Case 59 (4.7.27)

1. Each Entitys GameObject stored that was to be stored in EntityProvider.entityPool is checked against all the GameObjects in the scene
2. All GameObjects must match.

Result: Pass.

7.2.60 Test Case 60 (4.7.28)

1. ArgumentNullException is thrown.

Result: Pass.

7.2.61 Test Case 61 (4.8.1)

1. Entity.GetGameObject() returns GameObject equal to go.

Result: Pass.

7.2.62 Test Case 62 (4.8.2)

1. NullReferenceException is thrown.

Result: Pass.

7.2.63 Test Case 63 (4.8.3)

1. Entity.getName() returns Pokemon.

Result: Pass.

7.2.64 Test Case 64 (4.8.4)

1. NullReferenceException is thrown.

Result: Pass.

7.2.65 Test Case 65 (4.8.5)

1. The GameObjects renderers material color is blue.

Result: Pass.

7.2.66 Test Case 66 (4.8.6)

1. NullReferenceException is thrown.

Result: Pass.

7.2.67 Test Case 67 (4.8.7)

1. No exceptions are thrown by the system or the Unity game engine.

Result: Pass.

7.2.68 Test Case 68 (4.8.8)

1. NullReferenceException is thrown.

Result: Pass.

7.2.69 Test Case 69 (4.9.1)

1. Returns an instance of EntityFactory
2. Returns a CollectionFactory.

Result: Pass.

7.2.70 Test Case 70 (4.9.2)

1. Returns an instance of EntityFactory
2. Returns a CustomCollectionFactory.

Result: Pass.

7.2.71 Test Case 71 (4.9.3)

1. Returns an instance of EntityFactory
2. Returns a ModelFactory.

Result: Pass.

7.2.72 Test Case 72 (4.9.4)

1. Returns an instance of EntityFactory
2. Returns a LightFactory.

Result: Pass.

7.2.73 Test Case 73 (4.9.5)

1. Returns an instance of EntityFactory
2. Returns a ShapeFactory.

Result: Pass.

7.2.74 Test Case 74 (4.9.6)

1. Returns an instance of EntityFactory
2. Returns a ViewerFactory.

Result: Pass.

7.2.75 Test Case 75 (4.9.7)

1. Throws ArgumentNullException.

Result: Pass.

7.2.76 Test Case 76 (4.9.8)

1. Throws ArgumentException.

Result: Pass.

7.2.77 Test Case 77 (4.9.9)

1. Throws ArgumentException.

Result: Pass.

7.2.78 Test Case 78 (4.10.1)

1. Returns a List of strings, containing the lines of the CSV le.

Result: Pass.

7.2.79 Test Case 79 (4.10.2)

1. Throws ArgumentNullException.

Result: Pass.

7.2.80 Test Case 80 (4.10.3)

1. Throws FileNotFoundException.

Result: Pass.

7.2.81 Test Case 82 (4.11.1)

1. The light component of the Entitys GameObject returned is of type Spot.

Result: Pass.

7.2.82 Test Case 83 (4.11.2)

1. The light component of the Entitys GameObject returned is of type Area.

Result: Pass.

7.2.83 Test Case 84 (4.11.3)

1. The light component of the Entitys GameObject returned is of type Directional.

Result: Pass.

7.2.84 Test Case 85 (4.11.4)

1. The light component of the Entitys GameObject returned is of type Point.

Result: Pass.

7.2.85 Test Case 86 (4.11.5)

1. Throws LightTypeNotFoundException.

Result: Pass.

7.2.86 Test Case 87 (4.11.6)

1. Throws LightTypeNotFoundException.

Result: Pass.

7.2.87 Test Case 88 (4.11.7)

1. Throws InvalidListLengthException.

Result: Pass.

7.2.88 Test Case 89 (4.11.8)

1. Returns a basic Entity with a light component of type Spot.

Result: Pass.

7.2.89 Test Case 91 (4.11.9)

1. Returns a basic Entity with a light component of type Area.

Result: Pass.

7.2.90 Test Case 92 (4.11.10)

1. Returns a basic Entity with a light component of type Directional.

Result: Pass.

7.2.91 Test Case 93 (4.11.11)

1. Returns a basic Entity with a light component of type Point.

Result: Pass.

7.2.92 Test Case 94 (4.11.12)

1. Throws LightTypeNotFoundException.

Result: Pass.

7.2.93 Test Case 95 (4.11.13)

1. Throws ArgumentNullException.

Result: Pass.

7.2.94 Test Case 96 (4.11.14)

1. Throws ArgumentNullException.

Result: Pass.

7.2.95 Test Case 97 (4.12.1)

1. Returns an Entity instance.

Result: Pass.

7.2.96 Test Case 98 (4.12.2)

1. Throws InvalidListLengthException.

Result: Pass.

7.2.97 Test Case 99 (4.12.3)

1. Throws ArgumentNullException.

Result: Pass.

7.2.98 Test Case 100 (4.12.4)

1. Throws DirectoryNotFoundException.

Result: Pass.

7.2.99 Test Case 101 (4.12.5)

1. Throws FileNotFoundException.

Result: Pass.

7.2.100 Test Case 102 (4.12.5)

1. Throws FileNotFoundException.

Result: Pass.

7.2.101 Test Case 103 (4.13.1)

1. An Entity is returned
2. The Entity has a non-null GameObject within.

Result: Pass.

7.2.102 Test Case 104 (4.13.2)

1. ArgumentNullException is thrown.

Result: Pass.

7.2.103 Test Case 105 (4.13.3)

1. InvalidListLengthException is thrown.

Result: Pass.

7.2.104 Test Case 106 (4.14.1)

1. The object returned is not null
2. An Entitys GameObjects location is Vector3(8, 10, 12)
3. The Entity has a GameObject named ”RigidBodyFPSController”.

Result: Pass.

7.2.105 Test Case 107 (4.14.2)

1. ArgumentNullException is thrown.

Result: Pass.

7.2.106 Test Case 108 (4.14.3)

1. InvalidListLengthException is thrown.

Result: Pass.

7.2.107 Test Case 109 (4.14.4)

1. The object returned is not null
2. An Entity is returned
3. The Entity has a GameObject within, named ”RigidBodyFPSController”.

Result: Pass.

7.2.108 Test Case 110 (4.14.5)

1. ArgumentNullException is thrown.

Result: Pass.

7.3 Overview of Test Results

7.4 Functional Requirements Test Results

8 Other

9 Conclusions and Recommendations