|  |  |
| --- | --- |
| **Learner Name** | Husnain Ahmed |
| **Assessor Name** | Chris Livesey |
| **Qualification Title** | BTEC - L3 IT Extended Diploma in Computing |
| **Unit Title** | Unit 16 – Object Oriented Programming |
| **Assignment No./Title** | 16.2 – Design and Develop Object Oriented Solutions to Identified Problems |
| **Learning Aims** | B: Design object-oriented programming solutions to identified problems.  C: Develop object-oriented programming solutions to identified problems. |
|  |  |
| **Issue Date** | 19/04/21 |
| **Planned Submission Date** | 07/05/21 |
| **Re-submission Date (if approved)** | 21/05/21 |

|  |  |
| --- | --- |
| **Feedback** | Provided online in the form of rubric, comments, and general feedback. |
| **Reflection** | Once you receive feedback, you should reflect on your performance using the reflection document. This should be recorded on your Pro Portal.  --> ILP --> 3. My Learner Reflections |
| **SMART Actions/Targets** | You should regularly set SMART Actions/Targets on your Pro Portal.  --> ILP --> 4. My SMART Actions |

|  |  |  |
| --- | --- | --- |
| **Evidence of proof-reading and improvements to quality of communication using Microsoft Editor.** | | |
| Print screen of Microsoft Editor before improvements | Print screen of Microsoft Editor after improvements (1) | Print screen of Microsoft Editor after improvements (2)  Re-sub if required. |
|  |  |  |

Object-Oriented Project One

# Client Requirements

*With reference to the “Client Requirements” document, write a detailed report outlining how you will meet the requirements of the client.*

## Introduction and purpose

What does this document contain? What is the purpose of the task? What have you been asked to do? Who are you developing the game for? (Use the user requirements document)

|  |
| --- |
| This document contains evidence of the design process for my Escape game that I will be making in unit 16.2. The purpose of the task is to show progression in the unit and the design development. I am developing this game for all audiences.  The purpose of the game is to escape the rooms that you are locked in through opening doors and locks with keys. |

## Target Audience, Platform/Medium and programming languages

Age range, OS, programming language[s].

|  |
| --- |
| The target audience is anyone from ages 5 and upwards, regardless of gender. The reason for this is that the game is simple and addictive. As the game is very simple the players interests do not matter, depending on the chosen graphics you could cater to a certain audience, for example using cat sprites as the character to get animal lovers to play the game. I may choose to not use any graphics and make the game strictly through the use of CLI.  The target platform is windows and mac computers as a CLI game would not be suited to mobile devices.  This will be coded in C#. |

## Objectives

Main menu, high score etc. – should be based on the user requirements.

|  |
| --- |
| **Required Minimum Game Standard Objectives**   * There should be a welcome message and a short story behind the game. * There should be a main menu with the following options.   + Log in/Create new user.     - View user statistics.     - Play game. * There should be at least 3 levels (Rooms to escape from).   + Levels should increase in difficulty. * User details should be stored in an external file containing.   + Username, password, number of plays, average score. * The user should be able to create a new user account. * The user should be able to log in.   + The user should be given the option to view their statistics.   + The user should be able to play the game.     - There should be a first level.     - There should be a second level.     - There should be a third level.     - There should be a game completed message.     - There should be a scoring system.       * Point should be awarded for completing a level.       * Points should be deducted for making mistakes (mistakes do not always have to result in reduction of points).   **Desirable Objectives**   * Five levels of increasing difficulty. * An Option to save game progress. * A list of completed levels should show on the main menu. * The user has the option to choose to start from any previously completed level. |

## Development Methodology, Model and Time Scale

Choice of development methodologies and models with justification

|  |
| --- |
| The development methodology I have chosen to use to develop this program is the ‘Prototype methodology’, the reason for this is because I have already developed a prototype of this program that simply needs to be added to and completed, this will save me a lot of time. |

Gantt Chart

|  |
| --- |
|  |

# Program Structure Design

## Overview

Create a structure diagram to show the structure of the game. Justify your choice of structure.

|  |
| --- |
| **Initial Structure Diagram** |
|  |
| **Feedback for Optimisation.**  Feedback from two people:  Name of person providing feedback, List of possible improvements. |
| * Jay Chevli – Order the rectangles and add the extra level. * Chris Livesey – Add arrows and lines to show the order of the processes, also add the missing processes. |
| **Final Structure Diagram** |
|  |
| **Structure Justification**  Why is this structure diagram suitable? How does it meet the requirements? Why is it fit for purpose? |
| I have chosen to layout my structure diagram like this as it clearly shows the order of events in the game and what triggers lead to other specific events, due to this I can plan my game more easily and won’t accidentally leave out some events.  This follows all the objectives that the client requires such as the menu, game screen etc. (The name of the events are some of the objectives)  Once you open the app you will have to either log in or create an account, you will then be given a set of instructions before the game will start playing. You will then make your way through 4 different rooms before finishing the game. |

## Class Diagram

*Create a class diagram.*

|  |
| --- |
| **Initial Class Diagram** |
|  |
| **Feedback for Optimisation.**  Feedback from two people:  Name of person providing feedback, List of possible improvements. |
| * Jay Chevli – Add more details. * Chris Livesey – add details about the attributes and their relations with each other. |
| **Final Class Diagram** |
|  |
| **Class Diagram Justification**  Why is this class diagram suitable? How does it meet the requirements? Why is it fit for purpose? |
| The class diagram has tables for the Rooms, player, obstacles and items.  These are here to help me plan important functions that will be elaborated on in pseudocode and algorithms later, it also makes sure that I do not accidentally forget about some events and leave them out of the final game and cause errors.  It shows that the game will meet the client's requirements of having at least 3 levels to escape from. A way to log in with an account and a game over screen. |

## User Interface Plan/Designs

|  |
| --- |
| **UI Plan/Design**  Menu, Screen designs, characters/items/objects as appropriate |
|  |
| **Feedback for Optimisation.**  Feedback from two people:  Name of person providing feedback, List of possible improvements. |
| * Jay Chevli – Add more colour. * Chris Livesey – add colours that contrast to make it more aesthetically pleasing, due to the fact that you are using a CLI you cannot change much. |
| **Final UI Designs** |
|  |
| **User Interface Justification**  Why are these UI plans/designs suitable? How does it meet the requirements? Why is it fit for purpose? |
| These designs for the CLI user interface are suitable as they are very aesthetically pleasing and they can easily be distinguished, the choice of colours are easy to see and result in a satisfactory viewing experience for the player. |

## Algorithm Designs

*Use pseudocode to plan the algorithms for each of the objects that you identified in the class diagram – Use the table below you need a table for each class - add/delete rows as required.*

|  |  |
| --- | --- |
| **Class Name** | |
| * Room | |
| **Class Type** | |
| * Parent | |
| **Method name** | **Algorithm Design** |
| setUp Room | RoomName = "Room1"  RoomDescription = "This is a room"  LightStatus = "on" |
| getRoomName | RoomName = name |
| get roomDescription | RoomDescription = description |

|  |  |
| --- | --- |
| **Class Name** | |
| * LivingRoom | |
| **Class Type** | |
| * Child | |
| **Method name** | **Algorithm Design** |
| setUp LivingRoom | RoomName = "Store1"  RoomDescription = "This is a living room"  LightStatus = "off"  KitchenDoor1Status = "open"  ExitDoorStatus = "locked" |
| get/set Kitchen1DoorStatus | KitchenDoor1Status get set |
| get/set ExitDoorStatus | ExitDoorStatus get set |
| get/set lightStatus | LightStatus get set |

|  |  |
| --- | --- |
| **Class Name** | |
| * Kitchen | |
| **Class Type** | |
| * Child | |
| **Method name** | **Algorithm Design** |
| setUp Kitchen | RoomName = "Store1";  RoomDescription = "This is a kitchen";  LightStatus = "off";  StoreDoor1Status = "open";  StoreDoor1Status = "closed";  LivingRoomDoor1Status = "closed"; |
| get/set Store1DoorStatus | Store1DoorStatus get set |
| get/set Store2DoorStatus | Store2DoorStatus get set |
| get/set LivingRoom1DoorStatus | LivingRoom1DoorStatus get set |
| get/set lightStatus | lightStatus get set |

|  |  |
| --- | --- |
| **Class Name** | |
| * StoreRoom | |
| **Class Type** | |
| * Child | |
| **Method name** | **Algorithm Design** |
| setUp StoreRoom | RoomName = name  RoomDescription = description  LightStatus = light  DoorStatus = door |
| get/set doorStatus | doorStatus get set |
| get/set lightStatus | lightStatus get set |

|  |  |
| --- | --- |
| **Class Name** | |
| * Player | |
| **Class Type** | |
| * Parent | |
| **Method name** | **Algorithm Design** |
| setUp Player | UserName = name  Password = password  NumOfLogins = numOfPlays  Score = score |
| get PlayerDetails | PlayerDetails get |
| get UserName | UserName get |
| get Password | Password get |
| get/set Score | Score get set |

|  |  |
| --- | --- |
| **Class Name** | |
| * Item | |
| **Class Type** | |
| * Parent | |
| **Method name** | **Algorithm Design** |
| setUp Item | ItemName = "Item 1"  ItemDescription = "description" |
| get itemName | itemName get |
| get itemDescription | itemDescription get |

|  |  |
| --- | --- |
| **Class Name** | |
| * Key | |
| **Class Type** | |
| * Child | |
| **Method name** | **Algorithm Design** |
| setUp Key | ItemName = "Key 1"  ItemDescription = "This is a key that can be used to open a door"  Colour = "blue" |
| get colour | colour get |

# Test Plan

*Use the table to plan what tests will be needed.*

* *Test the functionality of the program - Think of tests that you can carry out to see if your system works.*
* *Remember to make use of normal, boundary and erroneous tests.*
* *Add more rows to the table - the aim is to test all the functions of the program.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Test Description**  *(What are you testing?)* | **Test data**  *(What data/inputs will you use?)* | **Test Type**  *(Normal, Boundary or erroneous?)* | **Expected Result** |
| 1 | If the program takes a string for the username | “jack” | Normal | It accepts the input |
| 2 | If the program takes a string for the password | “hello123” | Normal | It accepts the input |
| 3 | Do the menu options work correctly? | “1” | Normal | It accepts the input |
| 4 | Do the menu options work correctly? | "2” | Normal | It accepts the input |
| 5 | Do the menu options work correctly? | “79” | Erroneous | It does not accept the input and the program crashes |
| 6 | Do the menu options work correctly? | “Pan” | Erroneous | It does not accept the input and the program crashes |

# Implementation of Object-Oriented Requirements

*The following requirements must be used in at least one of your programs. Explain and give examples of how your solution has made use of each of the following:*

|  |  |  |
| --- | --- | --- |
| **OOP Requirement** | **Part of this program (Y/N)** | **Explanation and Example(s)** |
| Graphical User Interface | N | **GUI’s** have not been used, instead this program makes use of the command line interface (CLI) and does not show any images, instead being solely made up of text. This program does however make use of different colours for the text to distinguish it from others and to make it look more pleasing. |
| Inheritance | Y | **Inheritance** has been used in this program in the room class, the attributes of this class have been inherited into its subclasses ‘Kitchen’ and ‘LivingRoom’. The inherited attributes help save time and make the program more efficient as you can reuse the code instead of rewriting it all. |
| Polymorphism | Y | **Polymorphism** has been used for the object called ‘Key’ to change its properties at different times, it does this by changing its colour from red to blue to open different locks. This also makes the program more efficient and saves time by allowing this piece of code to be reused more than once. |
| Method Overloading | Y | **Overloading** has been used in the room class to change the value of the variables from strings to attributes. |
| Method Overriding | Y | **Overriding** has been used has been used to change the value of the method ‘getRoomDetails()’ in the room class, it updates the value of the light status and the door status as they have been changed from their previous state. |

# Implementation/Development

*The implementation evidence must show the use of Object-Oriented programming. Use the table structure below to show your program development. You will need a table for every object in your game.*

|  |
| --- |
| **Class/Object Name** |
| * ItemClass |
| **Class/Object Program Code** |
| using System;  using System.Collections.Generic;  using System.Text;    namespace EscapeGame16  {  class ItemClass  {  protected string ItemName { get; set; }  protected string ItemDescription { get; set; }    public ItemClass()  {  ItemName = "Item 1";  ItemDescription = "description";  }    public string getItemDetails()  {  return ("This is a " + ItemName + " " + ItemDescription + ".");  }      }    class Key : ItemClass  {  private string Colour { get; set; }    public Key()  {  ItemName = "Key 1";  ItemDescription = "This is a key that can be used to open a door";  Colour = "blue";  }    public Key(string name, string description = "This is a key that can be used to open a door", string colour = "red")  {  ItemName = name;  ItemDescription = description;  Colour = colour;  }    public string getKeyDetails()  {  return ("This is a " + Colour + " " + ItemName + ". " + ItemDescription);  }  }      } |

|  |
| --- |
| **Class/Object Name** |
| * PlayerClass |
| **Class/Object Program Code** |
| using System;  using System.Collections.Generic;  using System.Text;    namespace EscapeGame16  {  class PlayerClass  {  public string UserName { get; set; }  public string Password { get; set; }  private int NumOfLogins { get; set; }  private decimal Score { get; set; }    public PlayerClass(string name, string password, int numOfPlays = 1, decimal score = 0)  {  UserName = name;  Password = password;  NumOfLogins = numOfPlays;  Score = score;  }    public bool checkPassword(string inPassword)  {  if(inPassword == Password)  { return true; }  else  { return false; }  }    public decimal getScore()  {  return Score;  }    public void updateScore(int scoreIncrease)  {  Score = Score + scoreIncrease;  }    public void updateScore(decimal scoreIncrease)  {  Score = Score + scoreIncrease;  }    public void updateLogins()  {  NumOfLogins++;  }    public string getPlayerDetails()  {  return (UserName + " has logged in " + NumOfLogins.ToString() + " and has a score of " + Score);  }    }  } |

|  |
| --- |
| **Class/Object Name** |
| * RoomClass |
| **Class/Object Program Code** |
| using System;  using System.Collections.Generic;  using System.Text;    namespace EscapeGame16  {  class RoomClass  {  protected string RoomName { get; set; }  protected string RoomDescription { get; set; }  protected string LightStatus { get; set; }    public RoomClass()  {  RoomName = "Room1";  RoomDescription = "This is a room";  LightStatus = "on";  }    public RoomClass(string name, string description = "There is no description", string light = "on")  {  RoomName = name;  RoomDescription = description;  LightStatus = light;  }    public virtual string getRoomDetails()  {  return ("You are in " + RoomName + ". " + RoomDescription + ". The light is " + LightStatus + ".");  }    public string setLightStatus()  {  if (LightStatus == "off")  { LightStatus = "on"; }  else  { LightStatus = "off"; }    return ("The light is now " + LightStatus + ".");  }    }    class StoreRoom : RoomClass  {  private string DoorStatus { get; set; }    public StoreRoom()  {  RoomName = "Store1";  RoomDescription = "This is a storeroom";  LightStatus = "off";  DoorStatus = "closed";  }    public StoreRoom(string name, string description, string light = "off", string door = "closed")  {  RoomName = name;  RoomDescription = description;  LightStatus = light;  DoorStatus = door;  }    public string setDoorStatus()  {  if (DoorStatus == "closed")  { DoorStatus = "open"; }  else  { DoorStatus = "closed"; }    return ("The door is now " + DoorStatus + ".");  }    public override string getRoomDetails()  {  base.getRoomDetails();  return (RoomDescription + ". The light is " + LightStatus + "and there is a door that is " + DoorStatus + ".");  }    }    class Kitchen : RoomClass  {  private string StoreDoor1Status { get; set; }  private string StoreDoor2Status { get; set; }  private string LivingRoomDoor1Status { get; set; }      public Kitchen()  {  RoomName = "Store1";  RoomDescription = "This is a kitchen";  LightStatus = "off";  StoreDoor1Status = "open";  StoreDoor1Status = "closed";  LivingRoomDoor1Status = "closed";  }    public Kitchen(string name, string description, string light = "off", string sDoor = "open", string sDoor2 = "closed", string lDoor = "Locked")  {  RoomName = name;  RoomDescription = description;  LightStatus = light;  StoreDoor1Status = sDoor;  StoreDoor1Status = sDoor2;  LivingRoomDoor1Status = lDoor;  }    public string setStoreDoor1Status()  {  if (StoreDoor1Status == "closed")  { StoreDoor1Status = "open"; }  else  { StoreDoor1Status = "closed"; }    return ("The door is now " + StoreDoor1Status + ".");  }    public string setStoreDoor2Status()  {  if (StoreDoor2Status == "closed")  { StoreDoor2Status = "open"; }  else  { StoreDoor2Status = "closed"; }    return ("The door is now " + StoreDoor2Status + ".");    }    public string setLivingRoomDoorStatus( bool hasKey)  {  string returnString = "";  if (StoreDoor1Status == "closed")  {  returnString = (returnString + "A blue key is needed to open this door. ");  if (hasKey)  {  returnString = (returnString + "You have used the key to unlock the door. ");  StoreDoor1Status = "open";  }  }  else if (StoreDoor1Status == "closed")  { StoreDoor1Status = "open"; }  else  { StoreDoor1Status = "closed"; }    return ("The door is now " + StoreDoor1Status + "." + returnString);    }    }    class LivingRoom : RoomClass  {  private string KitchenDoor1Status { get; set; }  private string ExitDoorStatus { get; set; }      public LivingRoom()  {  RoomName = "Store1";  RoomDescription = "This is a living room";  LightStatus = "off";  KitchenDoor1Status = "open";  ExitDoorStatus = "locked";    }    public LivingRoom(string name, string description, string light = "off", string kDoor = "open", string eDoor = "Locked")  {  RoomName = name;  RoomDescription = description;  LightStatus = light;  KitchenDoor1Status = kDoor;  ExitDoorStatus = eDoor;  }    public string setExitDoorStatus(bool hasKey)  {  string returnString = "";  if (ExitDoorStatus == "locked")  {  if (hasKey)  {  returnString = (returnString + "You have used the key to unlock the door. ");  ExitDoorStatus = "open";  }  }  else if (ExitDoorStatus == "closed")  { ExitDoorStatus = "open"; }  else  { ExitDoorStatus = "closed"; }    return ("The door is now " + ExitDoorStatus + "." + returnString);  }      }        } |

|  |
| --- |
| **Class/Object Name** |
| * Program |
| **Class/Object Program Code** |
| using System;    namespace EscapeGame16  {  class Program  {  private static StoreRoom startingStoreRoom = new StoreRoom();  private static StoreRoom emptyStoreRoom = new StoreRoom("Store 2", "This is an empty store", "on", "open");  private static Kitchen theKitchen = new Kitchen("The Kitchen", "This is a kitchen. it has two storerooms and a locked door leading to the livingroom");  private static LivingRoom theLivingRoom = new LivingRoom("The Living room", "This is the Livingroom");    private static Key blueKey = new Key("Blue Key", "This key can be used to unlock a door", "blue");  private static Key redKey = new Key("Red Key");      static void PlayGame(PlayerClass player)  {  Console.Clear();  Console.ForegroundColor = ConsoleColor.DarkCyan;  Console.WriteLine("Escape Game");  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("Welcome, " + player.UserName + ", " + "You are starting with a score of : " + player.getScore().ToString());  Console.WriteLine("");  Console.WriteLine("you will need to escape this room");  Console.WriteLine("Escape the room!");  Console.WriteLine("");  Console.WriteLine(player.getPlayerDetails());    Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press enter to begin... ");  Console.ReadLine();  Console.ForegroundColor = ConsoleColor.White;  }    static void StartStoreRoom(ref PlayerClass player)  {  string userInput = "";  do  {  do  {  Console.Clear();  Console.ForegroundColor = ConsoleColor.DarkCyan;  Console.WriteLine("Escape Game: the unknown beginning");  Console.WriteLine(startingStoreRoom.getRoomDetails());  Console.WriteLine("");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to:");  Console.WriteLine(" [1] Run forward ");  Console.WriteLine(" [2] feel the walls for a light switch ");  userInput = Console.ReadLine();  if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  player.updateScore(-10);  Console.WriteLine("You run forward and bang your head, knocking yourself out");  Console.WriteLine("... you wake up some time later");  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  }  } while (userInput[0] != '2');    Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You found a switch on the wall - press enter to turn it on");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to turn on the switch");  Console.ReadLine();  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(startingStoreRoom.setLightStatus());  Console.WriteLine(startingStoreRoom.getRoomDetails());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to:");  Console.WriteLine(" [1] look at the door ");  Console.WriteLine(" [2] open the door ");  userInput = Console.ReadLine();  if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  player.updateScore(-10);  Console.WriteLine("While looking at the door you feel bored and fall asleep...");  Console.WriteLine("... you wake up some time later");  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  Console.WriteLine(startingStoreRoom.setLightStatus());  }  } while (userInput[0] != '2');  AreaComplete(ref player);  }    static void StartKitchen(ref PlayerClass player)  {  string userInput = "";  do  {  do  {  Console.Clear();  Console.ForegroundColor = ConsoleColor.DarkCyan;  Console.WriteLine("Escape Game: the kitchen");  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theKitchen.getRoomDetails());  Console.WriteLine("");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press enter to walk forward");  userInput = Console.ReadLine();    Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("As you walk forward a sensor activates...");  Console.WriteLine(theKitchen.setLightStatus());  Console.WriteLine("you can see a table with a small object on it");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to:");  Console.WriteLine(" [1] repeat the room details ");  Console.WriteLine(" [2] walk around the room ");  Console.WriteLine(" [3] walk over to the table and inspect the small object ");  Console.WriteLine(" [4] investigate what is inside the other room ");  userInput = Console.ReadLine();    if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theKitchen.getRoomDetails());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  }    else if (userInput[0] == '2')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theKitchen.getRoomDetails());  Console.WriteLine("you spend some time walking around the room");  Console.WriteLine("... eventually you stop and realise you are back where you started");  player.updateScore(-10);  theKitchen.setLightStatus();  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  }    else if (userInput[0] == '4')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("you enter the store room");  Console.WriteLine(emptyStoreRoom.getRoomDetails());  Console.WriteLine("... as the store room is empty you turn around and go back into the kitchen");  Console.WriteLine(theKitchen.setLightStatus());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  }    } while (userInput[0] != '3');    Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("you walk up to the table and inspect the object: ");  Console.WriteLine("...");  Console.WriteLine(blueKey.getItemDetails());  Console.WriteLine("...");    Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to:");  Console.WriteLine(" [1] repeat all of the room details so far");  Console.WriteLine(" [2] discard the item ");  Console.WriteLine(" [3] take the item and walk over to the locked door");  userInput = Console.ReadLine();    if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theKitchen.getRoomDetails());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  }    else if (userInput[0] == '2')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theKitchen.getRoomDetails());  Console.WriteLine("the key has been lost!");  Console.WriteLine("... you are unable to leave the room.");  Console.WriteLine("the room will be reset");  player.updateScore(-50);  theKitchen.setLightStatus();  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  }  } while (userInput[0] != '3');    Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("you walk over to the door. it has a blue lock");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to use the key in the lock");  Console.ReadLine();  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theKitchen.setLivingRoomDoorStatus(true));    AreaComplete(ref player);    }    static void StartLivingRoom(ref PlayerClass player)  {  string userInput = "";  do  {  Console.Clear();  Console.ForegroundColor = ConsoleColor.DarkCyan;  Console.WriteLine("Escape Game: The Living room");  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theLivingRoom.getRoomDetails());  Console.WriteLine("");    Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to:\n [1] Walk forward in the darkness?\n [2] Turn on the light switch?");  userInput = Console.ReadLine();  if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  player.updateScore(-10);  Console.WriteLine("You run forward and trip over a cardboard box, spraining your ankle and hitting your head on the ground, knocking yourself out...");  Console.WriteLine("... you wake up some time later\nYour score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to continue");  Console.ReadKey();  }  } while (userInput[0] != '2');    Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("Your hand is on a light switch");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to turn on the switch");  Console.ReadKey();  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theLivingRoom.setLightStatus());  Console.WriteLine(theLivingRoom.getRoomDetails());  Console.ForegroundColor = ConsoleColor.White;    do  {  Console.WriteLine("There is a chair with a table with a milkshake and tv remote on it infront of it.\nThere is a large flatscreen TV on the wall in front of the table and opposite the TV is a door.");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to:\n [1] repeat the room details?\n [2] walk around the room\n [3] sit on the chair\n [4] turn on the TV\n [5] look at the table\n [6] try to open the door");  userInput = Console.ReadLine();    if (userInput[0] == '1')  {  Console.WriteLine(theLivingRoom.getRoomDetails());  }    else if (userInput[0] == '2')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theLivingRoom.getRoomDetails());  Console.WriteLine("You walk around the room, fully exploring it");  Console.WriteLine("... eventually you stop and realise you are back where you began");  player.updateScore(-10);  theKitchen.setLightStatus();  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to continue");  Console.ReadKey();  }    else if (userInput[0] == '3')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You sit on the chair and feel that it is not resting on a flat surface");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to\n [1]stay sat down \n [2] stand back up\n [3] check under the chair");  userInput = Console.ReadLine();    if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You grow bored and you fall asleep");  player.updateScore(-10);  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to continue");  Console.ReadKey();  }    else if (userInput[0] == '2')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You stand up");  }  }    else if (userInput[0] == '4')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You turn on the tv with a remote that you found on the table\nThere is a news report which has a reporter talking about something however there is no sound.\nThe bottom headlines states a person has been missing for 5 days and is presumed dead\nThere is a picture of the person shown and it is you?");  }    else if (userInput[0] == '5')  {  do  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You look at the table\nThere is a milkshake sat on the table, will you drink it?\n [1] Yes\n [2] No");  if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You drink the milkshake, you feel your eyes getting heavy and you fall back onto the chair asleep.");  player.updateScore(-30);  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to continue");  Console.ReadKey();  }    else if (userInput[0] == '2')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You leave the milkshake");  }    } while (userInput[0] != '1' && userInput[0] != '2');  }    else if (userInput[0] == '6')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You find the door is locked and need to find a key");  }  } while (userInput[0] != '3');    Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You find a blue key");  Console.WriteLine(blueKey.getItemDetails());  Console.WriteLine("...");    do  {  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Would you like to:\n [1] repeat all of the room details so far?\n [2] discard the key?\n [3] take the key and walk over to the locked door?");  userInput = Console.ReadLine();    if (userInput[0] == '1')  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theLivingRoom.getRoomDetails());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to continue");  Console.ReadKey();  }    else if (userInput[0] == '2')  {  Console.ForegroundColor = ConsoleColor.Cyan;    Console.WriteLine(theLivingRoom.getRoomDetails());  Console.WriteLine("The key has been lost!");  Console.WriteLine("... you are unable to escape the room.\nThe room will be reset");  player.updateScore(-50);  Console.WriteLine("Your score is now: " + player.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to continue");  Console.ReadKey();  }    } while (userInput[0] != '3');    Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("You walk over to the door it has a blue lock");  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("Press any key to use the key in the lock");  Console.ReadKey();  player.updateScore(150);  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine(theLivingRoom.setExitDoorStatus(true));    AreaComplete(ref player);    }    static void AreaComplete(ref PlayerClass playerObject)  {  Console.ForegroundColor = ConsoleColor.Cyan;  playerObject.updateScore(10);  Console.WriteLine("The door opens and you walk out of the room");  Console.WriteLine("... well done you have escaped the room");  Console.WriteLine("Your score is now: " + playerObject.getScore());  Console.ForegroundColor = ConsoleColor.White;  Console.WriteLine("press enter to continue");  Console.ReadLine();  }    static void gameComplete(ref PlayerClass player)  {  Console.WriteLine("well done you have completed the game: ");  Console.WriteLine(player.getPlayerDetails());  Console.WriteLine("press enter");  Console.ReadLine();  }    static void Main(string[] args)  {  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("welcome to the escape game");  Console.WriteLine("");  Console.ForegroundColor = ConsoleColor.White;    string userName, password;  Console.WriteLine("Please enter your name");  userName = Console.ReadLine();  Console.WriteLine("Please enter your password");  password = Console.ReadLine();  Console.Clear();    PlayerClass player1 = new PlayerClass(userName, password, 1, 100);    PlayGame(player1);  StartStoreRoom(ref player1);  StartKitchen(ref player1);  StartLivingRoom(ref player1);  gameComplete(ref player1);    }  }  } |

# Testing

## System Testing

*This is where you test your program to show evidence that it works (or what does not work)*

* *Copy "Test Description", "Expected result" and Test Type from your test plan into the testing table below.*
* *Test the functionality of the program - Think of tests that you can carry out to see if your system works.*
* *Remember to make use of normal, boundary and erroneous tests.*
* *Add more rows to the table - the aim is to test all the functions of the program.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Test Description**  *(What are you testing?)* | **Expected Result**  *(What data/inputs will you use?)* | **Test Type**  *(Normal, Boundary or erroneous?)* | **Pass/Fail**  *If fail briefly describe remedial action* |
| 1 | If the program takes a string for the username | It accepts the input | Normal | Pass, it accepts the input |
| 2 | If the program takes a string for the password | It accepts the input | Normal | Pass, it accepts the input |
| 3 | Do the menu options work correctly? | It accepts the input | Normal | Pass, it accepts the input |
| 4 | Do the menu options work correctly? | It accepts the input | Normal | Pass, it accepts the input |
| 5 | Do the menu options work correctly? | It does not accept the input and the program crashes | Erroneous | Pass, as ‘79’ is not a valid input the prompt repeats itself |
| 6 | Do the menu options work correctly? | It does not accept the input and the program crashes | Erroneous | Pass, as ‘Pan’ is not a valid input the prompt repeats itself |

## Testing Print Screen Evidence

Each test must have one or more screen shots to show the outcome of the test. Where possible there should be a before and after screen shot

|  |  |  |
| --- | --- | --- |
| **Test** | **Print Screen 1** | **Print Screen 2**  (If required) |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |

## User Testing

*As two people of test your program. The aim is to gather some feedback to make some improvements to optimize the program. You will need to add 3 questions of your own that are specific to your program –* ***you MUST test each of the user requirement objectives.***

|  |  |  |
| --- | --- | --- |
| **Question** | **Tester 1** | **Tester 2** |
| **Name of tester?** | Jay Chevli | Chris Livesey |
| **Do all the menu options work?** | Yes. | Yes. |
| **Can you suggest any improvements to the menu?** | No. | Maybe add a GUI (Graphical User Interface) to make it more user friendly and aesthetically pleasing removing the need to type with clickable buttons. |
| **Is there a welcome message and a short story behind the game?** | No. | No, it simply says ‘You will need to escape the room, ESCAPE THE ROOM!’. It could be more detailed and informative. |
| **Can you log in / create a new user?** | Yes. | Yes. |
| **Are there at least 3 levels?** | Yes. | Yes. |
| **Do the levels increase in difficulty as they go on?** | Yes. | Yes. |
| **Are the user’s details stored in an external file? If so, can they be referred in later playthroughs of the game?** | No. | No, the user’s details are not stored in an external file and as such they cannot be referred to in later playthroughs of the game and the score cannot be saved to the user's account. |
| **Is the user given the option to view their statistics?** | No. | No, it should be implemented. |
| **Does the game work?** | Yes. | Yes, it does work. |
| **Is there a scoring system?** | Yes. | Yes. |
| **Can you suggest any improvements for the game?** | Add a welcome message / a short story behind the game before logging in. | Add the option to view the user's statistics before playing. |

# Optimization

You need to choose at least 2 optimizations and show evidence of implementing them.

|  |  |
| --- | --- |
| **Optimization Description** | Add a welcome message / a short story behind the game before logging in. |
| **Print Screen of development/Changes** | Console.WriteLine("Welcome to the escape game, You have awoken and found yourself trapped inside of a building! \n Try to escape the series of rooms you are in and get back home! \n There are 3 rooms that you need to find your way out of to leave the building! \n Have fun and try your best to survive!"); |
| **Print Screen showing it works (testing)** |  |

|  |  |
| --- | --- |
| **Optimization Description** | Add the option to view the user's statistics before playing. |
| **Print Screen of development/Changes** |  |
| **Print Screen showing it works (testing)** |  |

## Demonstration Video Link

[Microsoft Stream](https://web.microsoftstream.com/video/626be24d-f6a3-48ab-b25c-1c73dddcfdfd)



# Review/Evaluation

## Evaluation of the User Requirements

I have met most of the user requirements that were set earlier on in this document, these were:

The welcome message and a short story behind the game, this introduces you to the game and briefly explains the purpose of it as well as some lore to give more context to the game to interest the player. I have successfully implemented this.

The main menu with the options to view user statistics and play the game, this will allow the user to either view their current score or to play the game. I have successfully implemented this.

3 levels (rooms to escape from) that increase in difficulty, these 3 levels are named “kitchen”, “Storeroom” and “Living room” and get progressively harder for you to complete as you get further along. I have successfully implemented this.

Creating a new account and logging in, this should happen at the start of the program before the actual game starts, you will be given the option to enter a username and password, if the account has already been registered you will be able to log back into it. I have partially implemented this as my game allows you to create an account and play the game but not log back into a pre-existing account.

A game completed message, this is shown after the game and congratulates the user on completing the game. I have successfully implemented this.

A scoring system where points are deducted for making mistakes and granted for completing a level, either 10, 30 or 50 points are deducted for making a mistake. I have successfully implemented this.

The user requirements that I have not met are:

User details should be stored in an external file, The details such as username, password, score and the amount of games played are supposed to be contained here and shown to the user in the game when they log back in to their account. I have not added this function as there were too many bugs with its implementation which caused the program to crash.

## Strengths and Possible Improvements

One of the strengths of this game is that it uses CLI as opposed to GUI which reduces the file size. This means that it can be run on devices with small amounts of storage space. It can also be run on devices with lower specifications as it is a text based CLI which does not have very high visual requirements like a GUI would have due to the lack of images needing to be rendered. The only thing that would need time to render would be the coloured texts and even that extra rendering time is negligible due to how low it is.

Another strength of the game is its high degree of ease of useas there are instructions on how to navigate the program on the screen that are displayed to the user at the start of the program’s runtime, this ensures that they will know exactly how to use it. They are also given prompts of what to do at certain times such as ‘press enter to continue’ which make navigating the program much easier. Due to the differently coloured words, there is a clear distinction between the instructions of each room and the game itself, such as the options that you must choose from.

Another strength of the game is that it is very efficient as a lot of code can be reused due to it being inherited from parent classes, an example of this happening is in the room classes where a lot of attributes for the status of the room doors and lights are derived from the parent class of ‘room’. The fact that the console is cleared after each room is passed also shows efficiency as any data that is no longer necessary for the user, this gives a simplistic look that looks very efficient due to it not being confusing and distracting the user with superfluous amounts of information.

One of the weaknesses of this game is the fact that it uses a CLI as opposed to a GUI which makes it harsh on the eyes and not very interactive. This could be further improved upon by creating a GUI that would show a visual display of each room according to its description such as showing the doors and a table with an item on it in the rooms that they belong in.

This would increase the user’s experience when playing the game as it would be easier to visualise it when there is a picture given rather than going off a description. But this will increase the file size by a large amount which may make it harder to run, the use of a GUI would also make it harder to run as it Is not only rendering a series of text in the CLI but also images which is much more graphically intensive on a device.

Another weakness of the game is the lack of variety in its colour scheme, the use of less than 5 colours leaves much to me desired and can be harsh on the eyes when the user is bombarded with a block of long white text. This can be fixed by introducing multiple new colours into the fold such as yellow and neon pink.

Another weakness of the game is the fact that the users details and scores are not stored in an external file to be used in later playthroughs, due to this function not being introduced there is no save function in the game which could upset some more competitive users who may want to rack up a high score over multiple games.

## Evaluation of the chosen Methodology

The methodology that I have chosen is the Prototype methodology. The reason for this is because I have already developed a prototype of this program that simply needs to be added to and completed, this will save me a lot of time.

Using the prototype methodology, I only had to add the code for the 3rd level (The living room) and the 2 optimisations that I did.

This helped me save a lot of time as opposed to using the waterfall methodology and starting this program from scratch.

Object-Oriented Project Two

# Client Requirements

*With reference to the “Client Requirements” document, write a detailed report outlining how you will meet the requirements of the client.*

## Introduction and purpose

What does this document contain? What is the purpose of the task? What have you been asked to do? Who are you developing the game for? (Use the user requirements document)

|  |
| --- |
| This document contains evidence of the design process for my flappy bird styled game will be making in unit 14.2. The purpose of the task is to show progression in the unit and the design development. I am developing this game for all audiences.  The purpose of this game is to entertain the player and maintain competitiveness with a scoreboard to entice them to play more and get their friends into the game, the developers' purpose for making this game is to make money and gain renown for the game and themselves to get clout. |

## Target Audience, Platform/Medium and programming languages

Age range, OS, programming language[s].

|  |
| --- |
| The target audience is anyone from ages 5 and upwards, regardless of gender. The reason for this is that the game is simple and addictive. As the game is very simple the players interests do not matter, depending on the chosen graphics you could cater to a certain audience, for example using cat sprites as the character to get animal lovers to play the game.  The target platform is android as there are many phones that use android and there is a wider audience for it than IOS.  The languages that will be used to code this are C# and Unity. |

## Objectives

Main menu, high score etc. – should be based on the user requirements.

|  |
| --- |
| Main menu, GUI, Instruction screen, Game screen, game-over screen, a character than moves to the right constantly and is affected by gravity, score, scoreboard, game itself, choosing different sprite to play as (character selection), a touch screen function that works by tapping on the screen. |

## Development Model and Time Scale Gantt Chart

Choice of development methodologies and models with justification

|  |
| --- |
| The development methodology I have chosen to use to develop this program is the ‘Waterfall methodology’, the reason for this is because the program is relatively small in scale and can be completed in a simple linear fashion. |

Gantt Chart

|  |
| --- |
| C:\Users\b1905311\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\744352BF.tmp |

# Program Structure Design

## Overview

Create a structure diagram to show the structure of the game. Justify your choice of structure.

|  |
| --- |
| **Initial Structure Diagram** |
|  |
| **Feedback for Optimisation.**  Feedback from two people:  Name of person providing feedback, List of possible improvements. |
| * Jay Chevli – Order the rectangles. * Chris Livesey – Add arrows and lines to show the order of the processes. |
| **Final Structure Diagram** |
|  |
| **Structure Justification**  Why is this structure diagram suitable? How does it meet the requirements? Why is it fit for purpose? |
| I have chosen to layout my structure diagram like this as it clearly shows the order of events in the game and what triggers lead to other specific events, due to this I can plan my game more easily and won’t accidentally leave out some events.  This follows all the objectives that the client requires such as the menu, game screen etc. (The name of the events are some of the objectives)  Once you open the app the game will automatically start playing. Once you lose you will go to the main menu and be given the option to either replay the game, check the leader boards for your friends scores and your own, or to go to the store and use your acquired in game currency to buy new levels and character skins. |

## Class Diagram

*Create a class diagram.*

|  |
| --- |
| **Initial Class Diagram** |
|  |
| **Feedback for Optimisation.**  Feedback from two people:  Name of person providing feedback, List of possible improvements. |
| * Jay Chevli – Add more details. * Chris Livesey – add details about the attributes and their relations with each other. |
| **Final Class Diagram** |
|  |
| **Class Diagram Justification**  Why is this class diagram suitable? How does it meet the requirements? Why is it fit for purpose? |
| The class diagram has tables for the menu, background, playable character, obstacles, collectibles, score, game over screen, store and money.  These are here to help me plan important functions that will be elaborated on in pseudocode and algorithms later, it also makes sure that I do not accidentally forget about some events and leave them out of the final game and cause errors.  It shows that the game will meet the client's requirements of having a character that moves to the right constantly and is affected by gravity. A menu screen that leads to playing the game, a game over screen, objects that make you lose the game when you interact with them and much more.  The diagram also shows that when you come into contact with the coin object it will add to your coin counter and when you successfully pass by a pipe obstacle a total of 1 point is added to your score counter. |

## User Interface Plan/Designs

|  |
| --- |
| **UI Plan/Design**  Menu, Screen designs, characters/items/objects as appropriate |
| C:\Users\b1905311\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\CF6EC072.tmpC:\Users\b1905311\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2BDEABDE.tmpC:\Users\b1905311\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\96BD780A.tmpC:\Users\b1905311\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F09BC1C4.tmp |
| **Feedback for Optimisation.**  Feedback from two people:  Name of person providing feedback, List of possible improvements. |
| * Jay Chevli - Add more details to the background and colour in the character, make the pipe have special effects, maybe make the coin slightly transparent so that it doesn’t interrupt your game by blocking your vision. * Chris Livesey – Add more details to the assets for the characters, obstacles and background, maybe you could add some variations to them as well. |
| **Final UI Designs** |
|  |
| **User Interface Justification**  Why are these UI plans/designs suitable? How does it meet the requirements? Why is it fit for purpose? |
| 1st image - This is a generic city background with a road, buildings a cloudy sky and a small sun. The buildings are grey houses with differently designed and placed windows for variety. It meets the need of the client because it serves as a backdrop and the game screen. It is fit for its purpose because it gives off the idea of a city which is, but it may be too distracting and detailed.  2nd image - This design is of a tanned male man with a black hair and beard wearing a pair of black shoes, blue jeans and a white sleeveless vest. It meets the need of the client because it is a playable character. It is fit for its purpose because it is clearly distinguishable as a character and will not have any problems when moving around due to its small size in comparison to the obstacles, this character will move to the right constantly and is affected by gravity.  3rd image - This design is a grey door that has neon green liquid flowing through it in channels. I have emulated the effect of flowing liquid by having a green background and a blurred dark green jagged shape overlaying it. It meets the need of the client because it plays the role of an interactable object. It is fit for its purpose because it is the obstacle that can end the game when touching the player, this will also trigger the game over screen.  4th image - This is a slightly transparent gold coin; it is the in game acquirable currency. There are multiple coins in the game that you can pick up whilst playing. They are bronze, silver and gold. The gold coin is worth the most. It meets the need of the client because it plays the role of an interactable object. It is fit for its purpose because it does not distract from the overall gaming experience and obstruct your view if it is placed in the way of the character, this makes it so that the players gaming experience is not ruined. |

## Algorithm Designs

*Use pseudocode to plan the algorithms for each of the objects that you identified in the class diagram – Use the table below you need a table for each class - add/delete rows as required.*

|  |  |
| --- | --- |
| **Class Name** | |
| * CharacterMovement | |
| **Class Type** | |
| * Parent | |
| **Method name** | **Algorithm Design** |
| Start (start game) | Set time to 1  Initialise gravity with rigidbody component |
| Update (character moving) | Increase y coordinates by 5 when left click is pressed |
| Collision (with ground or pipe) | If in contact with column/pipe or ground  Set time to 0  Show gameover  Show menu  Append score to leaderboard |

|  |  |
| --- | --- |
| **Class Name** | |
| * Menu | |
| **Class Type** | |
| * Child | |
| **Method name** | **Algorithm Design** |
| ReplayGame | If clicking on replayButton in menu  Replay game  Set time to 1 |
| Store | If clicking on storeButton in menu  Close menu  Open store |
| Leaderboards | If clicking on scoreboardButton in menu  Close menu  Open Leaderboards |

|  |  |
| --- | --- |
| **Class Name** | |
| * Store | |
| **Class Type** | |
| * Child | |
| **Method name** | **Algorithm Design** |
| View money | Show collected number of coins as a text variable |
| Purchase background | Click to purchase if you have enough coins |
| Purchase character | Click to purchase if you have enough coins |
| Purchase pipe | Click to purchase if you have enough coins |

|  |  |
| --- | --- |
| **Class Name** | |
| * Leaderboards | |
| **Class Type** | |
| * Child | |
| **Method name** | **Algorithm Design** |
| View Scores | Show all scores |

|  |  |
| --- | --- |
| **Class Name** | |
| * Collectibles | |
| **Class Type** | |
| * parent | |
| **Method name** | **Algorithm Design** |
| Set position | Set x position of collectible  Set y position of variable |
| Hide upon touch | If player hitbox is touching collectible  Hide collectible |
| Collectible counter | If player hitbox is touching collectible  Add collectible to counter |
| Set size | Set size of collectible |

|  |  |
| --- | --- |
| **Class Name** | |
| * Coin | |
| **Class Type** | |
| * child | |
| **Method name** | **Algorithm Design** |
| Collect coin | If player hitbox is touching coin  Coin counter = coin counter + 1 |

# Test Plan

*Use the table to plan what tests will be needed.*

* *Test the functionality of the program - Think of tests that you can carry out to see if your system works.*
* *Remember to make use of normal, boundary and erroneous tests.*
* *Add more rows to the table - the aim is to test all the functions of the program.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Test Description**  *(What are you testing?)* | **Test data**  *(What data/inputs will you use?)* | **Test Type**  *(Normal, Boundary or erroneous?)* | **Expected Result** |
| 1 | Is the character effected by gravity? | Start the game and watch to see if the character falls. | Normal | Yes, the character should be affected by gravity. |
| 2 | Does the game over screen show when the character hits the floor / a pipe? | Start the game and direct the character into the floor or a pipe. | Normal | Game over screen shows. |
| 3 | Does the score counter go up by passing a pipe? | Start the game and direct the character through a pipe, then check the score counter. | Normal | The score counter should go up by one. |
| 4 | Does the character jump/move when you tap on the screen? | Start the game, then tap on the screen. | Normal | Character will jump when you tap on the screen. |
| 5 | What happens if you die before getting any score? | Start the game, fall on the floor or crash into the first pipe. | Boundary | Game-over screen shows a score of 0. |
| 6 | Does swiping the screen have the same effect as tapping it? | Start the game, swipe your finger on the screen. | Erroneous | It will count as a single jump, similar to tapping on the screen. |
| 7 | Does the start game button work? | Start the game. | Normal | IT should work and start the game. |
| 8 | Does the replay button work? | Start the game, die, tap the replay button. | Normal | It should work and replay the game. |
| 9 | Does your high score show on the leader boards? | Start the game, die, tap the leader boards button. | Normal | Your score will show where you are on the leader boards compared to the top scorers. |
| 10 | Can you buy new character models on the in-game shop with in-game currency? | Start the game, die, tap the shop button, tap the character model you want to buy. | Normal | You can buy new character models as long as you have enough in-game currency. |
| 11 | Does the game over screen show when you die? | Start the game, die. | Normal | It should show up. |

# Implementation of Object-Oriented Requirements

*The following requirements must be used in at least one of your programs. Explain and give examples of how your solution has made use of each of the following:*

|  |  |  |
| --- | --- | --- |
| **OOP Requirement** | **Part of this program (Y/N)** | **Explanation and Example(s)** |
| Graphical User Interface | Y | I have used a GUI in the game to show the character, background and the obstacles. |
| Inheritance | Y | The code from the pipe / column object has been inherited by the column spawner variable that spawns multiple columns as obstacles for the player. |
| Polymorphism | Y | The pipe / column object has used polymorphism by to spawn multiple columns as obstacles for the player. |
| Method Overloading | N | It is not used. |
| Method Overriding | N | It is not used. |

# Implementation/Development

*The implementation evidence must show the use of Object-Oriented programming. Use the table structure below to show your program development. You will need a table for every object in your game.*

|  |
| --- |
| **Class/Object Name** |
| * StarCharacter |
| **Class/Object Image** |
|  |
| **Class/Object Program Code** |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.SceneManagement; // this is importing the needed directories I will be using    public class StarCharacter : MonoBehaviour  {  public float Speed;  Rigidbody2D rb;    public GameObject ReplayButton;  public Score ScoreText;  public GameObject ScoreButton;  public GameObject StoreButton;  public GameObject Menu;  public GameObject GameOver; //initialising all the variables to be used that correlate to the components in unity    // Start is called before the first frame update  void Start()  {    Time.timeScale = 1;  rb = GetComponent<Rigidbody2D>();    }    // Update is called once per frame  void Update()  {  if (Input.GetMouseButtonDown(0))  {  rb.velocity = Vector2.up \* Speed;  } // moves the character up when the left mouse button is clicked to simulate jumping    }    private void OnTriggerEnter2D(Collider2D collision) // this function will be activated if the component that has been designated as a trigger comes into physical contact with a hitbox  {  if (collision.CompareTag("Column")) // this tag is for the empty space (the aforementioned hitbox) after a pipe if the character goes past it this if statement is triggered  {  print("Score Up"); //outputs “Score Up” to the console  ScoreText.ScoreUp(); //increases the value of the ScoreText component by 1 and visibly shows this on the score counter  }  }    private void OnCollisionEnter2D(Collision2D collision) // this function will be triggered if the character collides with something which will be set later  {  if (collision.gameObject.CompareTag("Ground") ||  collision.gameObject.CompareTag("Pipe")) //this if statement is triggered if the character comes into contact with what has been designated as the ground or the pipes  {  Time.timeScale = 0; // pauses the game by setting the time to 0  Menu.SetActive(true); // sets the menu to active so that it shows and can be interacted with  ReplayButton.SetActive(true); // sets the replay button to active so that it shows, if clicked on the game will be reset  GameOver.SetActive(true); // sets the game over image to active so that it shows  ScoreButton.SetActive(true). // sets the score button to active so that it shows, when clicked on the leader boards will be shown to the user  StoreButton.SetActive(true); // sets the store button to active so that it shows, if clicked on the store will be opened so that the player can buy new skins to use in the game with in-game currency  }    }    public void ReplayGame() // this code will be stored as a function called ‘ReplayGame’  {  SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex); //this is a function that will reset the game when the button that it is assigned to is clicked on, it does this by loading the starting scene of the game being played    }  } |

|  |
| --- |
| **Class/Object Name** |
| * Ground / pipes (collision) |
| **Class/Object Image** |
|  |
| **Class/Object Program Code** |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine; // this is importing the needed directories I will be using    public class Ground : MonoBehaviour  {  public float Speed;  float GroundLength;    BoxCollider2D GroundCollider;      public float DestroyXPlace; //initialising all the variables to be used that correlate to the components in unity    // Start is called before the first frame update  void Start()  {  if(gameObject.CompareTag("Ground")) // Any component that has been tagged as “ground” such as the floor will trigger this if statement  {  GroundCollider = GetComponent<BoxCollider2D>(); //The length of the ‘BoxCollider2D’ component will be set as the value of the variable “GroundCollider”  GroundLength = GroundCollider.size.x; // this sets the value of the variable ‘GroundLength’ to be the same as the size of the variable ‘GroundCollider’  }    }    // Update is called once per frame  void Update()  {  transform.position = new Vector2(  transform.position.x - Speed \* Time.deltaTime,  transform.position.y); // this changes the position of the ground component by moving it ‘speed’ amount of places to the left, the higher the value of the variable ‘speed’ the more paces to the left the ground will move per second and the faster it will appear to move    if (gameObject.CompareTag("Ground")) // Any component that has been tagged as “ground” such as the floor will trigger this if statement  {  if (transform.position.x <= -GroundLength) // this if statement will be triggered if the value of ‘transform.position’ is less than or equal to the ‘groundLength’  {  transform.position = new Vector2(  transform.position.x + 2 \* GroundLength,  transform.position.y); // this makes it so that once the ground has moved off of the screen (twice its own length) it will be moved to the other side of the screen so that it can continue moving to the left and being an obstacle for the character, if this did not happen the character would fall through the floor instead of colliding with it.  }  }    if (gameObject.CompareTag("Column")) //this if statement will be triggered by any components that have been tagged as ‘column’, in specific this is referring to the pipes that have moved off screen and contains the code needed to remove these objects once they have moved off of the screen so that they do not take up too much memory in the ram and cause the game to lag  {  if (transform.position.x <= -DestroyXPlace) //this if statement will be triggered when the value of ‘DestroyXPlace’ (that you will set in unity) is more than or equal to the value of ' transform.position.x’  {  Destroy(gameObject); // if the aforementioned conditions have been met the object tagged as ‘column’ will be destroyed’  }  }      }  } |

|  |
| --- |
| **Class/Object Name** |
| * Column spawner |
| **Class/Object Image** |
|  |
| **Class/Object Program Code** |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine; // this is importing the needed directories I will be using    public class ColumnSpawner : MonoBehaviour  {  public GameObject ColumnPrefab;  public float MinY, MaxY;    float timer;  public float MaxTime; //initialising all the variables to be used that correlate to the components in unity    // Start is called before the first frame update  void Start()  {  SpawnColumn(); // the ‘SpawnColumn’ function will be triggered on game start  }    // Update is called once per frame  void Update()  {  timer += Time.deltaTime; // this sets the timer to timer + Time.deltaTime which is the time it takes for each frame to render  if(timer >= MaxTime) // This if statement will be triggered when the value of ‘MaxTime’ is less than or equal to the value of the timer  {  SpawnColumn(); //triggers the ‘SpawnColumn’ function and spawns a column  timer = 0; // sets the timer to 0 and pauses the game  }  }    void SpawnColumn() // this code will be stored as a function called ‘SpawnColumn’  {    float RandomYPosition = Random.Range(MinY, MaxY); // This allows you to set the value of ‘MinY’ and ‘MaxY’ in unity as well as setting them as the two extremes for the range of where the columns can be spawned within  GameObject NewColumn = Instantiate(ColumnPrefab);  NewColumn.transform.position = new Vector2(  transform.position.x,  RandomYPosition); // This spawns in the column in that aforementioned range  }  } |

|  |
| --- |
| **Class/Object Name** |
| * score |
| **Class/Object Image** |
|  |
| **Class/Object Program Code** |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI; // this is importing the needed directories I will be using    public class Score : MonoBehaviour  {    int score; //initialising the variable that will be used that correlate to the components in unity  // Start is called before the first frame update  void Start()  {  score = 0; // sets the score to 0 at the start of the game and resets it to 0 when replaying the game  }    // Update is called once per frame  public void ScoreUp() // this code will be stored as a function called ‘ScoreUp’  {  score++; //increases the value of the ScoreText component by 1  GetComponent<Text>().text = score.ToString(); // visibly shows the increase of the ‘ScoreText’ variable on the score counter    }  } |

|  |
| --- |
| **Class/Object Name** |
| * Menu |
| **Class/Object Image** |
|  |
| **Class/Object Program Code** |
| (this code is in the end of the StarCharacter script, as of now only the code for the replay button is working)  private void OnCollisionEnter2D(Collision2D collision) // this function will be triggered if the character collides with something which will be set later  {  if (collision.gameObject.CompareTag("Ground") ||  collision.gameObject.CompareTag("Pipe")) //this if statement is triggered if the character comes into contact with what has been designated as the ground or the pipes  {  Time.timeScale = 0; // pauses the game by setting the time to 0  Menu.SetActive(true); // sets the menu to active so that it shows and can be interacted with  ReplayButton.SetActive(true); // sets the replay button to active so that it shows, if clicked on the game will be reset  GameOver.SetActive(true); // sets the game over image to active so that it shows  ScoreButton.SetActive(true). // sets the score button to active so that it shows, when clicked on the leader boards will be shown to the user  StoreButton.SetActive(true); // sets the store button to active so that it shows, if clicked on the store will be opened so that the player can buy new skins to use in the game with in-game currency  }    }    public void ReplayGame() // this code will be stored as a function called ‘ReplayGame’  {  SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex); //this is a function that will reset the game when the button that it is assigned to is clicked on, it does this by loading the starting scene of the game being played    }  } |

# Testing

## System Testing

*This is where you test your program to show evidence that it works (or what does not work)*

* *Copy "Test Description", "Expected result" and Test Type from your test plan into the testing table below.*
* *Test the functionality of the program - Think of tests that you can carry out to see if your system works.*
* *Remember to make use of normal, boundary and erroneous tests.*
* *Add more rows to the table - the aim is to test all the functions of the program.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Test Description**  *(What are you testing?)* | **Expected Result**  *(What data/inputs will you use?)* | **Test Type**  *(Normal, Boundary or erroneous?)* | **Pass/Fail**  *If Fail briefly describe remedial action* |
| 1 | Is the character effected by gravity? | Start the game and watch to see if the character falls. | Normal | pass |
| 2 | Does the game over screen show when the character hits the floor / a pipe? | Start the game and direct the character into the floor or a pipe. | Normal | pass |
| 3 | Does the score counter go up by passing a pipe? | Start the game and direct the character through a pipe, then check the score counter. | Normal | pass |
| 4 | Does the character jump/move when you tap on the screen? | Start the game, then tap on the screen. | Normal | pass |
| 5 | What happens if you die before getting any score? | Start the game, fall on the floor or crash into the first pipe. | Boundary | Pass, you just die with a score of 0 nothing special occurs |
| 6 | Does swiping the screen have the same effect as tapping it? | Start the game, swipe your finger on the screen. | Erroneous | Fail, this cannot be tested as unity uses the mouse which cannot simulate the swiping motion on a touchpad, but I assume that it will work |
| 7 | Does the start game button work? | Start the game. | Normal | Fail, I have chosen to remove the need for a start game button and instead have the game automatically start upon launch, extra games can be played through tapping on the replay button. |
| 8 | Does the replay button work? | Start the game, die, tap the replay button. | Normal | Pass |
| 9 | Does your high score show on the leader boards? | Start the game, die, tap the leader boards button. | Normal | Fail, clicking on the button does not have any effect and does not redirect you to the leader boards as they have not been coded yet |
| 10 | Can you buy new character models on the in-game shop with in-game currency? | Start the game, die, tap the shop button, tap the character model you want to buy. | Normal | Fail, clicking on the button does not have any effect and does not redirect you to the shop as it has not been coded yet |
| 11 | Does the game over screen show when you die? | Start the game, die. | Normal | Pass |

## Testing Print Screen Evidence

Each test must have one or more screen shots to show the outcome of the test. Where possible there should be a before and after screen shot

|  |  |  |
| --- | --- | --- |
| **Test** | **Print Screen 1** | **Print Screen 2**  (if required) |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |

## User Testing

*As two people of test your program. The aim is to gather some feedback to make some improvements to optimize the program. You will need to add 3 questions of your own that are specific to your program –* ***you MUST test each of the user requirement objectives.***

|  |  |  |
| --- | --- | --- |
| **Question** | **Tester 1** | **Tester 2** |
| **Name of tester?** | Abdul Razzaq | Hasib Ahmed |
| **Do all of the menu options work?** | No | Not as of yet |
| **Can you suggest any improvements to the menu?** | Making all of the buttons work | Ensuring that all of the functions on the main menu work such as the scoreboard and the in-game store |
| **Were there any problems with the game itself?** | No | No |
| **Can you suggest any improvements to the character?** | You could make it so that it makes noises when moving | Maybe animate it to give it some special effects |
| **Can you suggest any improvements for the GUI?** | Add more colours to the menu buttons. | Add a different design for the menu. |
| **Does the character than moves to the right constantly, and if so, can you suggest any improvements?** | Yes, make the game progressively faster after each pipe. | Yes, change the speed every 10 pipes to be faster or slower. |
| **Is the character affected by gravity, and if so, can you suggest any improvements?** | Yes, an improvement could be to increase the gravity to make the game harder. | Yes, an improvement to the game could be to alter the gravity acting on the character withing a range after it passes through each pipe so the player will be kept on their toes due to the varying gravity. |
| **Were there any problems with the score counter?** | No, but you can change the font to something fancier to grab the player’s eyes, however this may distract them from the game and cause issues. | No, but you could improve it by changing the colour of it to make it more aesthetically pleasing and stand out as white is very plain. |
| **Does the touch screen function work?** | yes | yes |
| **Can you choose what sprite to play as?** | No | Not as of yet as the store function does not work. |
| **Does the shop work properly?** | no | no |
| **Does the score board work properly?** | no | no |
| **Can you suggest any improvements for the game?** | Make the score board work properly. | Add coins so that they can be collected and used in the in-game store |

# Optimization

You need to choose at least 2 optimizations and show evidence of implementing them.

|  |  |
| --- | --- |
| **Optimization Description** | Making the leader boards work |
| **Print Screen of development/Changes** | Currently the ‘scores’ button does nothing. I will make it redirect the player to a blank leaderboard    Now I will make headings for the leaderboard    Now I will add a replay button so you can play the game again and make the heading a bit bigger to make them more easily readable    Now i will add in some placement values    I have changed my mind and decided that I will instead use another scene with a black background that automatically redirects you to a scoreboard |
| **Print Screen showing it works (testing)** |  |

|  |  |
| --- | --- |
| **Optimization Description** | Making the in game store work as well as adding coins to the game to be used in the store |
| **Print Screen of development/Changes** | I will spawn in coins within a range    I will add a coin counter in the top right    I will now collect the coins    The coins do not disappear when you replay the game so they can be used in the shop after multiple tries      I will make it so that clicking on the store button changes the background to a store    I will now add images of what the buyable options are and the button to buy them |
| **Print Screen showing it works (testing)** | I will now test it    The game ends instantly as the character is spawning into a pipe that is left over from the last game.    I fixed this issue by spawning the next character slightly in front of the previous one, so it does not get stuck in the pipe |

## Demonstration Video Link

<https://web.microsoftstream.com/video/bc82780f-e24f-4c1e-bde3-051014bd2772>

[Microsoft Stream](https://web.microsoftstream.com/video/bc82780f-e24f-4c1e-bde3-051014bd2772)



# Review/Evaluation

## Evaluation of the User Requirements

I have met the user requirements that were set earlier on in this document, these were:

Main menu, this function appears to the user once the character dies by hitting the pipe or floor.

GUI, this was shown in the menu with buttons that redirect you or do other functions.

Instruction screen, this was not shown at all as I have forgot it and once the game is started it automatically starts making your character move, hue to the ease of control in this game with only clicking on the screen this shouldn’t be too hard to figure it out.

Game screen, this can be seen as soon as the game has begun as it automatically is shown, it is in the background in the menu and the store and is shown once again once you click on the replay button.

Game-over screen, I do not have a game over screen but a game over image that is shown underneath the menu once you lose, this has the same effect as a game over screen and is an adequate replacement for it.

A character than moves to the right constantly and is affected by gravity, this was achieved by having the floor, coins and obstacles move to the left to give off the visual effect of the character moving to the right and past the background. The character is also affected by gravity due to the RigidBody2D component that automatically made this happen.

Score, the score is shown on the top of the game screen as you are playing and increases as you go past each obstacle.

Scoreboard, this has been called ‘leaderboards’ and is a button in the main menu that can be accessed once you die. In this function you can enter your name and score into the text fields, and they will be appended to the leaderboards in the correct order.

Choosing different sprite to play as (character selection), this has been done by having one sprite that is automatically chosen for you and another that is an optional choice in the store for you to play with once you have collected enough coins.

A touch screen function that works by tapping on the screen, this is the way that the game will be played on the PC this is replaced with the use of the left mouse click that can make the character jump into the air.

## Strengths and Possible Improvements

One of the strengths of the final program is that it only has 1 button that is needed to play the game, so it is very easy to understand and play.

Another strength is that the menu is easy to navigate and is very simplistic in both design and function.

Another strength is that the coin is see through and does not distract you from the game.

One of the weaknesses of the final project is that the coin can sometimes spawn in places that are too far away from the gap in the pipe which makes it hard to get without dying. It can also spawn directly inside of the pipes which makes it nigh impossible to collect, I have attempted to improve this by making the movement speed of the coins slower than that of the pipes and making them spawn closer to the middle of the screen on the y axis so that it doesn’t make it so that the user must go all the way to the top or bottom to collect them.

Another weakness is that there is no top or ‘lid’ to the game, the character can escape the confines of the game area without dying. This makes it so that the character is not on screen and could cause possible bugs. This can be improved by adding an invisible lid to the top that ends the game if the character comes into contact with it similar to the floor.

Another weakness is that the hitbox of the car shaped pipes is rectangular while the cars themselves are not, this makes it so there is some blank space that is counted as a hitbox which would dissatisfy the user if they lost the game due to hitting an invisible wall that they could not see. This can be improved by making the hitbox properly fit the irregular shape of the car by using a different component than BoxCollider2D that isn't restricted to rectangular shapes.

## Evaluation of the chosen Methodology

The methodology that I had chosen to use to create this program was the ‘Waterfall methodology’. This was used as this was a small-scale project with a single person working on it. This game was not that complicated and did not need to be very detailed, so the waterfall method was a good choice.

Using the waterfall methodology, I coded this program in a linear fashion and completed each stage in order. I used a YouTube tutorial and coded the program in unity according to it using a step-by-step approach.

If I had instead used the RAD methodology the user interface would have been better, but that methodology is suited to larger scale projects with more people working on it, so it was not suitable for this project.

# Evidence of Individual Responsibility

I have shown individual responsibility by designing the OOP escape game by myself and doing the 3rd level without any external help.

I managed my time effectively according to the Gantt chart that I made very early on in this project, I completed the project before the deadline due to the project already being partially completed and due to using the prototype methodology.

I have shown individual responsibility in solving problems in this project by receiving feedback from 2 different people and working on that feedback to optimise the program even further by adding an option to view the users' statistics before and after the game, I also added a welcome message and a short backstory before the game to introduce the user to the game and briefly explain the purpose of it as well as some lore to give more context to the game to interest the player.

I have shown a high level of creativity in coming up with a scenario for the backstory of the game and for the 3rd level of the game.

I behaved appropriately in my conduct whilst undertaking this project and met deadlines on time and some even before the time. I have showed responsibility and leadership in helping others by explaining what some of the tasks were when they were unaware of the contents of the project and were stuck on certain areas such as coding or coming up with scenarios for their 3rd level of the game.

Some improvements could be made to this project, I have mentioned these earlier on in my evaluation above.

I have shown individual responsibility by designing all of the flappy bird game myself without any external help.

I have shown individual responsibility in managing my time effectively according to my Gantt chart that I designed earlier on. All the deadlines for the separate parts of my project were met on time and achieved to my utmost capabilities.

I have shown individual responsibility in solving problems that occurred in this project by receiving feedback from 2 different people and working on that feedback to optimise the program even further by adding a coin counter, shop and leader board. This makes it so that there are more functions than just the game itself.

I have shown high levels of creativity by designing all of my assets and graphics that I have used in this project by myself without any external help. My best design that shows the most creativity was the pipes that resemble cars due to their strange shape as well as the design for the alien background and pipes due to the strange grey and green colour scheme and glowing neon green areas within the pipe.

I behaved appropriately in my conduct whilst undertaking this project and met deadlines on time. I have showed responsibility and leadership in helping others by explaining what some of the tasks were when they were unaware of the contents of the project and were stuck on certain areas such as coding or adding components to unity.

Some improvements could be made to this project, I have mentioned these earlier on in my evaluation above.

|  |
| --- |
| **Witness Statement**  This student has demonstrated individual responsibility, creativity and self-management at each of the stages of this project |
| **Tutor Signature/Digital Signature:** |