**Software Engineering**

**Design Document**

**Birds of Prey**

Team 6 Members:

Sebastian Farias - Team Leader and Technical Writer

Maaz Ansari - Configuration Manager

Anibal Ruiz - API - Liaison

Jean Claude Charles Boute - Webmaster

**Table of Contents**

1. Introduction

1.1 Purpose of the System

1.2 Design Goals

1.2.1 Correctness

1.2.2 Consistent

1.2.3 Complete

1.2.4 Robust

1.2.5 Usability

1. High-Level Software Architecture

2.1 Subsystem Decomposition

2.2 Hardware/Software Mapping

2.3 Persistent Data Management

2.4 Access Control and Security

2.5 Boundary Conditions

1. Subsystem Services

3.1 User Interface

3.2 Scenes

3.3 Model

1. Low-Level Design

4.1 Overview

4.2 Final Object Design

4.3 Class Interfaces

4.3.1 Main Menu

4.3.2 How to Play

4.3.3 Credits

4.3.4 Load Level

4.3.5 Pause Menu

4.3.6 Game Over Menu

4.3.7 Button and Text

4.3.8 Levels

4.3.9 Level Controller

4.3.10 Red\_Bird

4.3.11 Enemy

4.3.12 Crate

4.4 Packages

1. Glossary and References
2. Team Member Contributions
3. **Introduction**

**1.1 Purpose of the System**

Our game “Birds of Prey”, is a simple 2-D single player game in which any person of any age can play. The user interface will be simple to navigate as it will all have clear button layouts, and the instructions and controls of the game will be simple as well. It will be playable on all desktop and laptop devices by downloading the program. The story and setting of our game is that thieves have invaded the forest habitat of the birds and it is up to the birds to destroy all of the thieves and their tower structures in order to take back the land that is rightfully theirs. To complete the game, the player controls the playable birds and must launch them and destroy the thieves and their tower structures for each level. Our game will be fast, easy to navigate, and fun to play. The player can play the game as many times as they want and can navigate through our entire game simply with a mouse or a laptop trackpad.

**1.2 Design Goals**

As a team, we must ensure that every part of our project is as clear and consistent as can possibly be. Our design goals are as follows:

**1.2.1 Correctness**

Our final deliverable must be able to satisfy all of the requirements that we have stated in our analysis document. For example, the playable birds must be able to launch and eliminate enemies.

**1.2.2 Consistent**

We will make our game consistent; every level that we make will have little to no bugs and will be easily manageable, as well as the menus including the main menu and the pause menu.

**1.2.3 Complete**

Our game must be complete; this means that our game must be fully functional. All features of our game must be fully functional, for example: The main menu of the game must be fully functional, all levels of the game must work and behave as we have intended them to behave, and our pause menu must behave as a mean to stop the game and give the user options to resume, restart the level, or quit the game.

**1.2.4 Robust**

We will make sure that our game has minimal to no bugs so the user can have a pleasant and easy user experience playing our game.

**1.2.5 Usability**

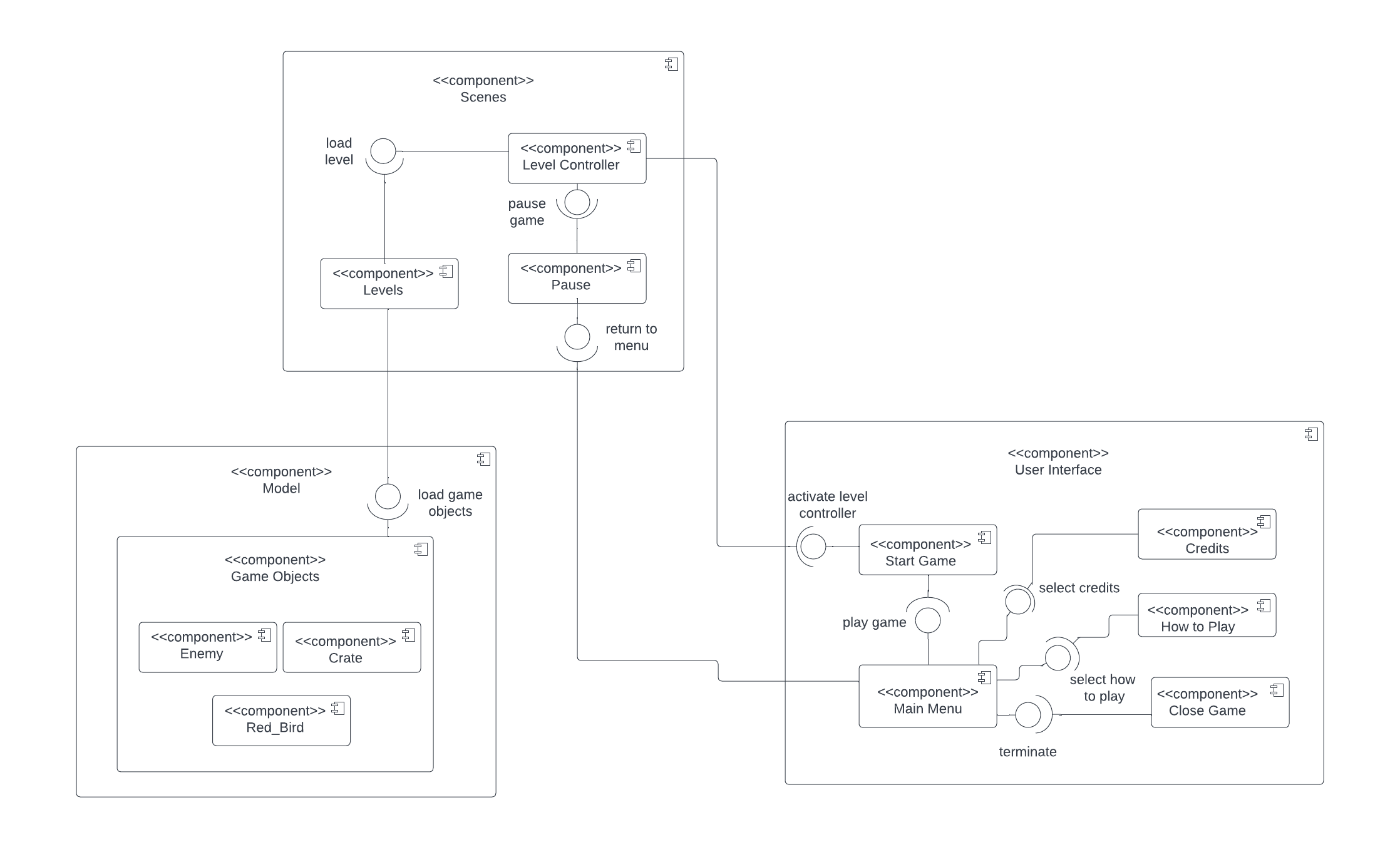
The user interface of our game must be foolproof and easy to comprehend so anyone of any age can easily play and have a fun experience.

1. **High-Level Software Architecture**

Overview:

We have introduced the proposed software architecture of our game project in the following subsections. First, subsystem decomposition of packages and every single interaction is demonstrated. In section 2, we will explain the high-level software architecture of our system; this will consist of subsystem decomposition, hardware/software mapping and persistent data management. We would like to design the game in a way that the implementation processes would be easier, intuitive and straightforward; so, in section 3 we will break down the subsystem decomposition and explain the components of the system. Finally, section 4 of the design document will include our final object design diagram which will include all of the classes and objects we intend to implement in our system.

**2.1 Subsystem Decomposition**



For this specific section, we want to decompose our system into subsystems. During the decomposition, our purpose is to reduce a lot of connections between different subsystems of the main system and increase coherence of the components. By decomposing the game system as described we can easily modify the game or extend it when it is needed.

During the decomposition of the system, we have decided that the Model-View Controller system design pattern is a good fit to apply to our program. We divided our system based on MVC principles.

* **Our system has three subsystems**

We have a **user interface subsystem** which includes some components and the menu interface for the player.

We have the **model subsystem** which includes the game objects. Some components as enemies, the bird and crates.

We have the **scenes subsystem** which includes the controller of the system levels and the pause menu.

* Dependencies and patterns between classes can be represented easily in this model.
* Model-view controller system is an effective design pattern which will increment the efficiency of the implementation stage.

We divided our system into three subsystems. These three subsystems are User Interface subsystem, Scenes subsystem and Model subsystem. We tried to decompose these systems according to their different functionalities. On the other hand, each subsystem has a responsibility to invoke other systems, so that the game can stay maintainable during runtime.

* **The User Interface subsystem** can only request a functional action to the Scenes subsystem by requesting ‘level controller’ to Game Scenes component of Scene Subsystem. Therefore any interaction between User Interface subsystem and Game scene subsystem must occur through level controller components. We will make sure that there will be flawless communication between the user and decision support system
* **Model Subsystem**, includes components for game entities which are basically game objects. According to the data coming from the controller they update their views in the UserInterface subsystem.
* **The Scene Subsystem** acts like a controller of the game. Basically it has some components:
  + The level controller, which controls the game, pauses the game, and is connected to the menu.
  + It has all the levels of the game.

We will go more in detail in the next chapter.

**2.2 Hardware/Software Mapping**

The programming language that we will use for implementing the core design of our game project is C# and the game engine used for our game is Unity. Our game runs on one PC at a time. (No multiplayer option and no network connection feature.) Software can be executed with executable files among other files.

**2.3 Persistent data management**

Birds of Preydoes not require any complex data storage system or database. Our game will store the game instances, like objects, maps, and scenes in the hard drive of the player. Having a game that keeps session to session storage is more complex; what we want is to be able to create a game that is easy to use without having to access an external database.

**2.4 Access control and security**

Birds of prey will not use internet or network connection or any database to operate. It is simple, you start the software, load the game, choose the start game option and you can start enjoying the game; anybody can play the game. Therefore, there will be no access control and security requirements to play. The user experience will be unique and there will be no need to save; the player will just try to pass every level of the game.

**2.5 Boundary Conditions**

**2.5.1 Initialization**

The user just needs to load the game and the main menu will display all of the options that the user can choose from which will include the start game button and the close game button.

**2.5.2 Termination**

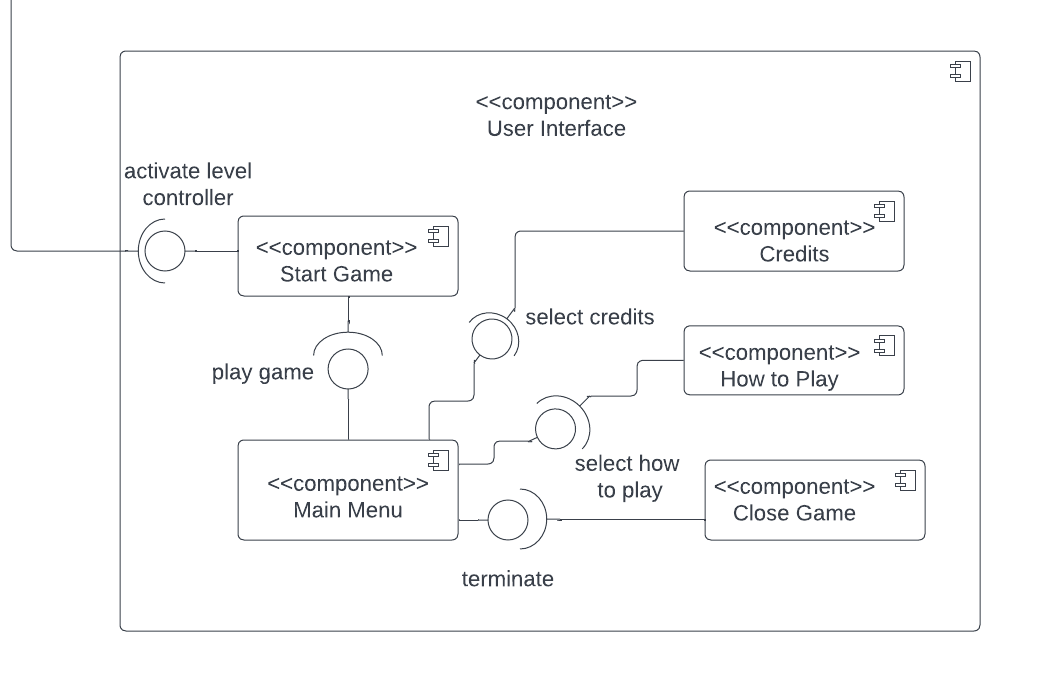
The user can terminate the game by simply choosing the close game option from the main menu and the game will be terminated.

**2.5.3 Failure**

If the program cannot continue executing then the user can terminate the game and then restart it if there are any issues.

1. **Subsystem services**

**3.1 User Interface**

****

User Interface subsystem is responsible for providing user interface for Birds of prey. It has five major components which are:

1. Main Menu

2. Start Game

3.How to Play

4. Credits

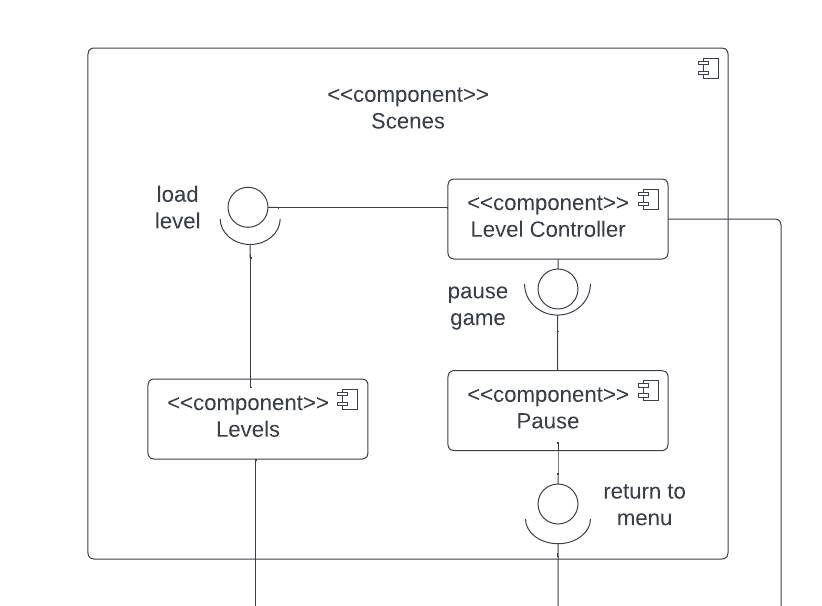
5. Close Game

When the user first starts the game, the user interface provides a menu through its Menu component. Menu component has classes for different types of menus; some of these menus will include the Pause Menu and the Stop Menu. Each of them has different responsibilities for the user interface subsystem.

Most importantly, only through the Menu component can a new game be initialized. The Main Menu components can be invoked by stop/exit game signals coming from Scene subsystem components. The Credits component will be invoked from the Menu component.

Lastly, the How to play component guides the player through the main concepts of the game, what to do, and what you have to do to win or pass the levels; it will guide you through all of the necessary info you need to play this game. The Credits component will display all the credits and references used to build the game. The Close Game will be invoked through the Menu component, This is used when you want to exit the game.

**3.2 Scenes**

****

The scene subsystem is responsible for controlling and handling the game. Some of the components of the scene subsystem include the level controller, the load level, the levels, pause game, and return to menu. The level controller controls the game loop of Birds of Prey.

Major components:

1. Level Controller
2. Load level
3. Levels
4. Pause Game
5. Return to menu

**Load level**: loads the current level the player wants to play.

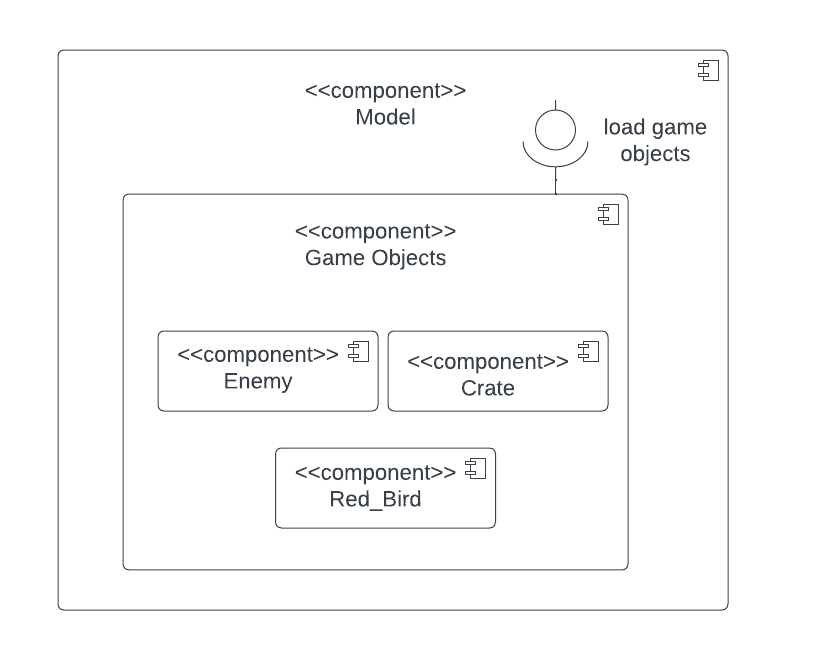
**Level controller:** controls the game. It is responsible for pausing the game, starting the game, and loading levels once the previous level is completed.

**Pause the game**: The player has the option to pause the game to stop playing.

**Levels:** includes all the levels we have designed for the game.

**Return to menu:** is an option if the player wants to return to the main menu.

**3.3 Model**

****

**Model Subsystem** represents our models. It consists of our game objects which can be different types. Model has a main component called GameObjects component which represents different types of game objects/entities (bird, enemies, crates). Basically, all game entities can only be updated and drawn by the method call coming from the Scene subsystem. Model class has one component which includes the following two subcomponents:

Game Objects:

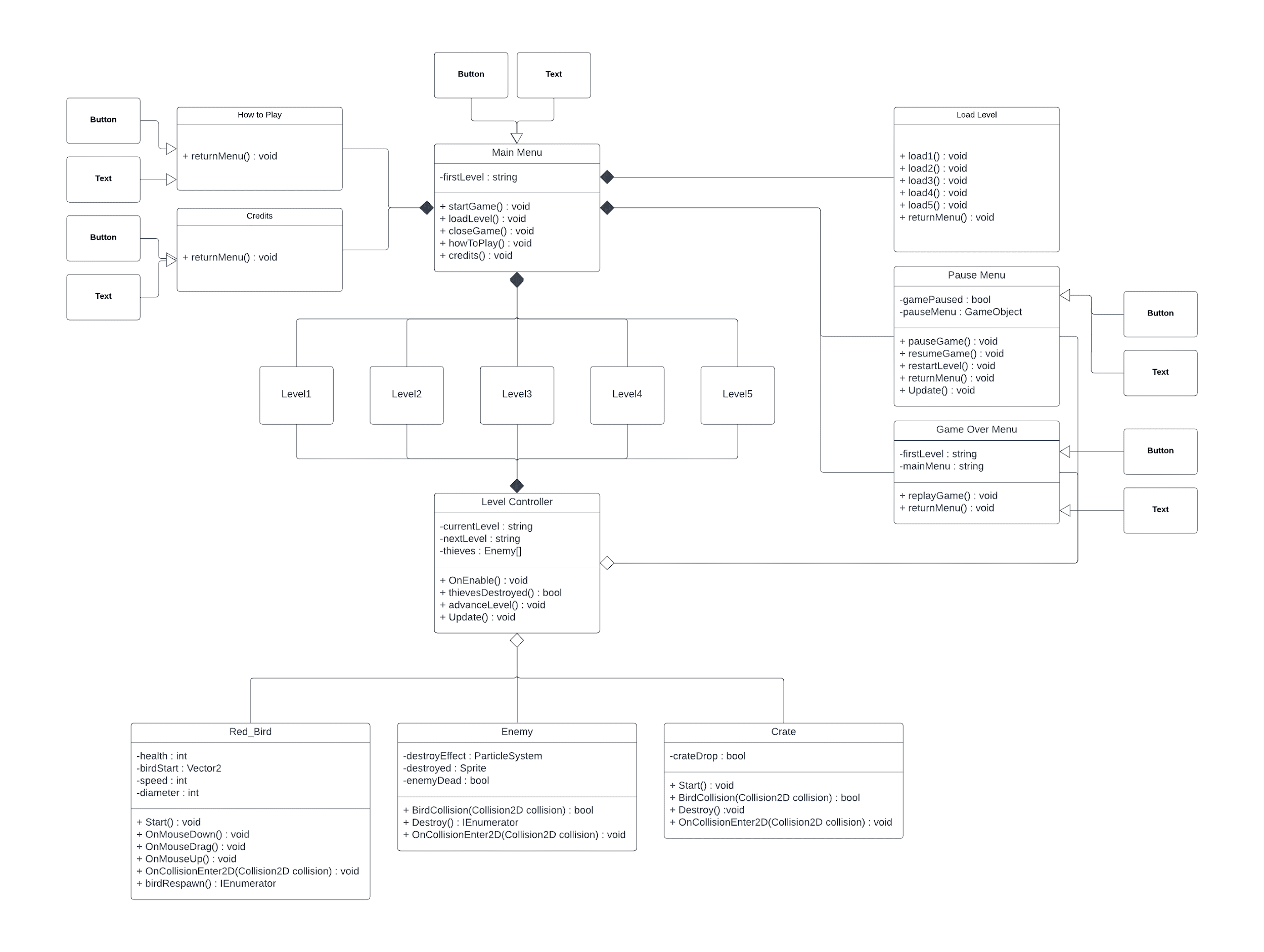
* Enemy: Represents the antagonists of the game. In order to win the game, the players must destroy the enemies to win.
* Crate: Obstacles surrounding the enemies. Must be destroyed by the bird to catch enemies.
* Red\_Bird: Character used by the player throughout the game. This is the playable character that the user utilizes.
* According to the control flow coming from controller model updates, it is viewed by calling proper methods which make changes in the View subsystem.

1. **Low-Level Design**

**4.1 Overview**

In section 4 of the design document, we will break down the entirety of our final object design. We will include a class diagram of all of the objects we intend to implement in our program and we will specify how every attribute and method will behave. We intend to make our program reliable and easy to maintain; the user interface will also be easy to navigate, so the user can easily be able to pick it up with minimal knowledge or experience using and playing computer games.

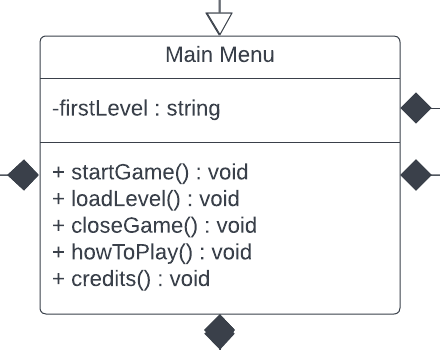
**4.2 Final Object Design**

****

**4.3 Class Interfaces**

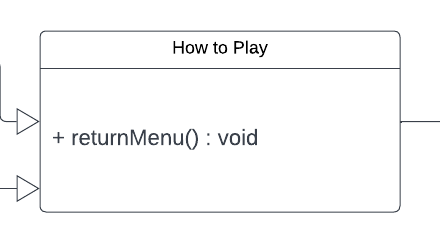
In this section we will describe all of the objects that we plan to implement in our program and what the behavior of each function is as well as the attributes.

**4.3.1 Main Menu**

****

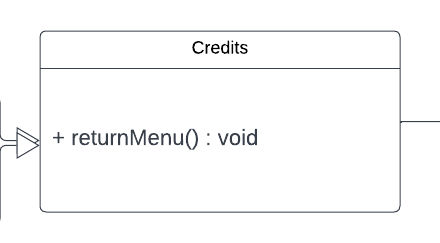
* firstLevel : string - this attribute will allow the first level of the game to be loaded when the user presses the “Start Game” button.
* startGame() : void - this function will start the game and load the very first level.
* loadLevel() : void - this function will take the user to a screen that will give them the option to choose any level that they wish to play. The user must press the “Load Level” button from the main menu.
* closeGame() : void - this function will terminate the game when the user presses the “Close Game” button.
* howToPlay() : void - this function will take the user to a screen that will display a text box of the instructions of the game as well as the controls of the game. The user must press the “How to Play” button in order to get to this screen.
* credits() : void - this function will take the user to a screen that will display a text box of the developer credits of the game as well as the references we have used in order to develop the game. The user must press the “Credits” button in order to get to this screen.

**4.3.2 How to Play**

****

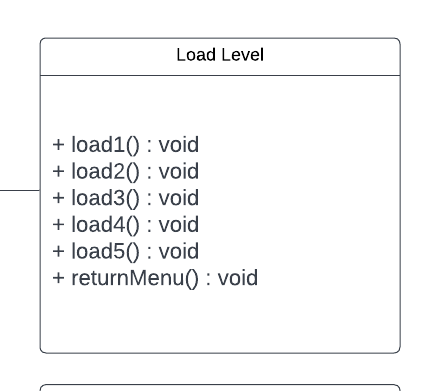
* returnMenu() : void - this function will allow the user to return to the main menu from the “How to Play” screen. The button to return to the main menu will say “Return to Main Menu”.

**4.3.3 Credits**

****

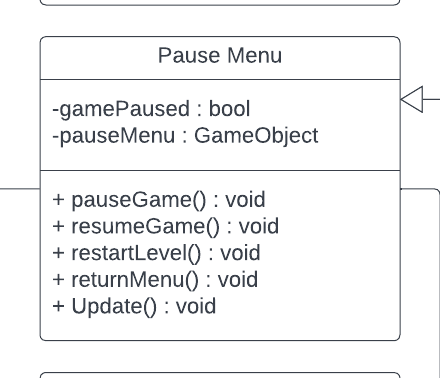
* returnMenu() : void - this function will allow the user to return to the main menu from the “Credits” screen. The button to return to the main menu will say “Return to Main Menu”.

**4.3.4 Load Level**

****

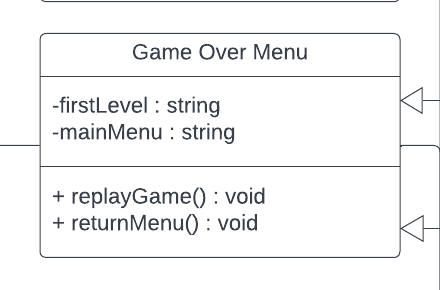
* load1() : void - this function will load the first level of the game; the user must select the “Level 1” button which is accessed from the “Load Level” screen.
* load2() : void - this function will load the second level of the game; the user must select the “Level 2” button which is accessed from the “Load Level” screen.
* load3() : void - this function will load the third level of the game; the user must select the “Level 3” button which is accessed from the “Load Level” screen.
* load4() : void - this function will load the fourth level of the game; the user must select the “Level 4” button which is accessed from the “Load Level” screen.
* load5() : void - this function will load the fifth level of the game; the user must select the “Level 5” button which is accessed from the “Load Level” screen.
* returnMenu() : void - this function will allow the user to return to the main menu from the “Load Level” screen. The button to return to the main menu will say “Back”.

**4.3.5 Pause Menu**

****

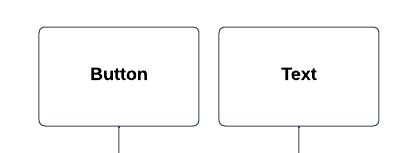
* gamePaused : bool - this attribute is a boolean value, and it is set as false.
* pauseMenu : GameObject - this attribute is a GameObject value; once it is set active it will display the pause menu.
* pauseGame() : void - this function will set the pauseMenu game object active; it will then freeze the current frame of the game to keep it from running in the background, and will set the gamePaused boolean value to true.
* resumeGame() : void - this function will release the pauseMenu game object; it will then continue to run the game from the frame in which it was paused, and then it will set the gamePaused boolean value to false.
* restartLevel() : void - this function will reload the current level in which the user was currently playing; it will reload all birds, enemies, and crates to their original position.
* returnMenu() : void - returnMenu() : void - this function will allow the user to return to the main menu from the “Pause Menu” screen. The button to return to the main menu will say “Return to Main Menu”.
* Update() : void - this is a default method in C# and Unity. This function runs frame by frame and the function of it in our pause menu will be to check if the user has paused the game; it will also check if the user has chosen to resume the game. The user can pause the game and resume the game by pressing ‘p’ on their keyboard.

**4.3.6 Game Over Menu**

****

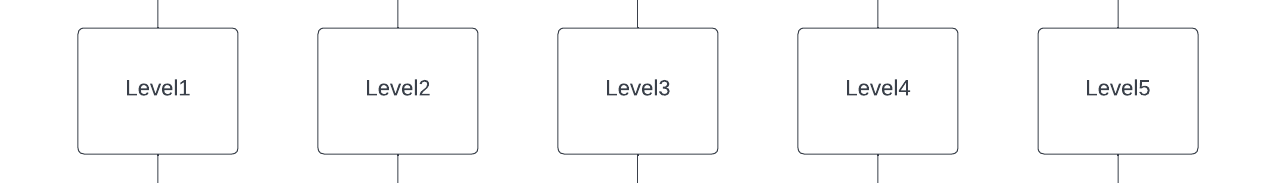
* firstLevel : string - this attribute is given to the first level of the game, so the system can load the first level of the game.
* mainMenu : string - this attribute is given to the main menu, so the system can load the “Main Menu '' screen.
* replayGame() : void - this function will restart the game for the user; it will take the firstLevel string value which will then load the scene of “Level1”.
* returnMenu() - this function will allow the user to return to the main menu from the “Game Over” screen. The button to return to the main menu will say “Return to Main Menu”.

**4.3.7 Button and Text**



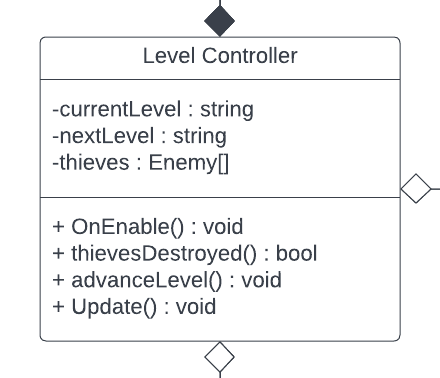
* Button and text are interfaces that will be shown on every menu option such as the main menu, pause menu, game over menu, how to play menu, and credits menu. Text will display a text box, and button will allow the player to navigate throughout the game.

**4.3.8 Levels**

****

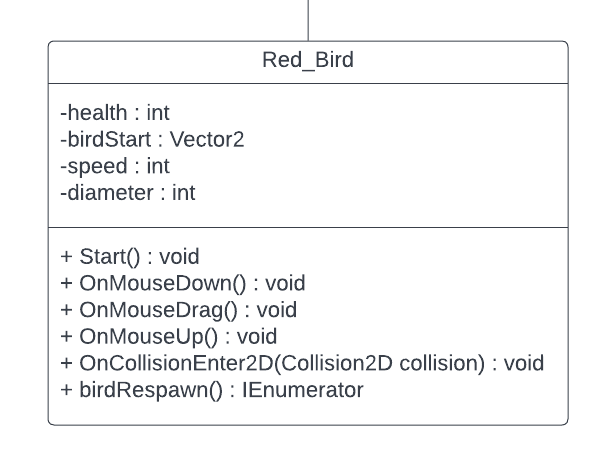
* These are all 5 levels of the game; starting the game will load level one, and the user can have access to play each level from the “Load Level” screen. Every level has a different amount of thieves for the user to defeat as well as different and unique tower structures that are defending the thieves.

**4.3.9 Level Controller**

****

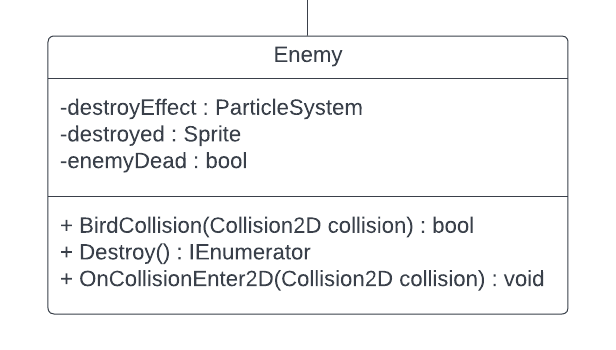
* currentLevel : string - this attribute takes the name of the current level and puts the value into a string data type.
* nextLevel : string - this attribute takes the name of the next level and puts the value into a string data type allowing the level controller to access and load the next level.
* thieves : Enemy[] - this attribute takes all of the thieves that were loaded into the level and stores it as a game object.
* OnEnable() : void - this function will be active and run for the entirety of the level; it will keep track of all of the thieves or enemies that are still alive by using the Enemy[] thiefs attribute.
* thievesDestroyed() : bool - this boolean function will return either a true or false value. If there are thieves remaining on the level, then the boolean value will return false; however, if all thieves are destroyed then the function will return true as a value.
* advanceLevel() : void - this function will advance to the next level of the game. It will use the nextLevel string value and generate the next level.
* Update() : void - this is a default method in C3 and Unity. It will run frame by frame and for the Level Controller object it will run a conditional statement or an if statement. If the thiefsDestroyed() method returns true (which means that there are no enemies remaining in the level), then it will activate the advanceLevel() method.

**4.3.10 Red\_Bird**

****

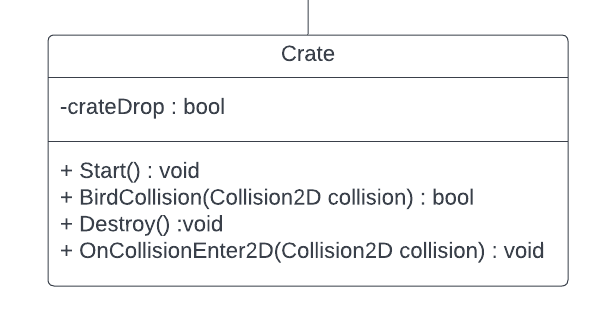
* health : int - this attribute stores the health of the bird which can be changed by the player to make it lower or higher.
* birdStart : Vector2 - this attribute represents the position of the bird on the 2D grid; the value is stored in birdStart to be able to use it for other functions in other methods.
* speed : int - this attribute stores the integer value in which the speed of the bird will be launched; this attribute can also be changed by the user to make the bird fly faster or slower.
* diameter : int - this attribute stores the integer value of how far back the bird can be pulled back for launch; this attribute can also be changed by the user to pull back the bird farther or not as far in order to determine the power and distance in which the bird can be launched.
* Start() : void - this function will be executed as soon as the game starts; the bird game object will return the grid position in which the bird is placed and store it into birdStart, and the bird object will be set to kinematic which means that the bird will stay in that position until it is moved by the user.
* OnMouseDown() : void - this function will activate when the user clicks and holds on the bird by using their mouse or trackpad, it will then turn the bird object a different color to let the user know it is activated.
* OnMouseDrag() : void - this function will activate when the user clicks and holds the bird object and then drags (or pulls back) the bird object. When the bird is dragged, it will take the starting position of the bird (birdStart : Vector2) and the position on the grid in which it was dragged to; it will then change the trajectory of the bird depending on the position the user drags the bird to on the grid. Once the user lets go of the mouse click, it will launch the bird in the direction that the user has aimed.
* OnMouseUp() : void - this function will activate once the user lets go of the initial mouse click after dragging and powering up the bird. This will power up the bird and launch the bird in the direction in which it was aimed by the user.
* OnCollisionEnter2D(Collision2D collision) : void - once the bird object is launched by the user and it collides with another object on the same level (either an enemy or a crate), it will then call the birdRespawn function.
* birdRespawn() : IEnumerator - this function will tell the system to wait a set amount of seconds before the bird respawns in the position where it was initially set when the level was loaded. This function is called by OnCollisionEnter2D(Collision2D collision) which is called after the bird object collides or makes contact with another game object.

**4.3.11 Enemy**

****

* destroyEffect : ParticleSystem - the destoryEffect attribute stores the particle effect system from unity.
* destroyed : Sprite - the destroyed attribute is a sprite attribute that will change the image of the enemy once it is destroyed.
* enemyDead : bool - this attribute stores a boolean value if the enemy is dead.
* BirdCollision(Collision2D collision) : bool - this function will return a boolean value of true or false, and it will detect if the bird game object has collided with the enemy game object. If the bird has collided with the enemy at a certain point, then it will return true and the enemy is destroyed.
* Destroy() : IEnumerator - this function will destroy or get rid of the enemy sprite on the screen. Once destroyed, the sprite will change to the dead sprite image, the particle effect will play (destroyEffect), and after a few seconds the enemy will disappear from the screen.
* OnCollisionEnter2D(Collision2D collision) : void - this function will call the previous functions mentioned. If the bird collides with an enemy it will return true on the BirdCollision function, then it will call the Destroy() function and begin the process of said function.

**4.3.12 Crate**

****

* crateDrop : bool - this attribute is a boolean value that will return either a true or a false value.
* Start() : void - this function will be executed once the game level loads and generates. It will set the kinematic state to true which means that the crate will stay in the place in which it was positioned until it collides with another object.
* BirdCollision(Collision2D collision) : bool - this function will return a boolean value of true or false, and it will detect if the bird game object has collided with the crate game object. If the bird has collided with the crate at any point, then it will return true.
* Destroy() : void - this function will execute if the crateDrop boolean is true; it will set the kinematic state of the crate to false and let it fall with gravity.
* OnCollisionEnter2D(Collision2D collision) : void - this function behaves similarly to the OnCollisionEnter2D function on the enemy object. If the bird makes contact with a crate, then the BirdCollision function will return true; once it returns true then it will call the Destroy() method and cause the crates to fall.

**4.4 Packages**

* using System.Collections; - this package provides a multitude of data structures such as coroutines.
* using System.Collections.Generic; - this package provides general interfaces, classes, and other collections.
* using UnityEngine; - this package tells the system the language C# is being used to create the program.
* using UnityEngine.SceneManagement; - this package allows the program to switch between menu scenes by a click command, and switches between level scenes once the user completes a level by eliminating all of the enemies.

1. **Glossary and References**

[**https://docs.unity3d.com/Manual/index.html**](https://docs.unity3d.com/Manual/index.html)

**https://www.youtube.com**

[**https://www.lucidchart.com/**](https://www.google.com/aclk?sa=l&ai=DChcSEwj6ks7pyJT3AhWIn7MKHZy7AJEYABAAGgJxbg&ae=2&sig=AOD64_34jbCSqVX4ikfw_tQKOKJB-Hje7g&q&adurl&ved=2ahUKEwi0iMTpyJT3AhVehIkEHWTsDmAQ0Qx6BAgDEAE)

1. **Team Member Contributions**

**Sebastian Farias** - Team Leader and Technical Writer:

My contribution to the team’s design document was section 1 and 4, as well as the component diagram used in section 2.1 and section 3. Section 1 of the design document gives an overview of our project and the goals we have to implement in our software program; we want our game to be as clean and complete as possible, and for the user to have an easy and fun experience playing our game. The component diagram used in section 2 and section 3 was created using Lucid Chart and it helps our team break down the components we must implement for our game; I created the diagram and my team helped to write the description and to analyze all the aspects of the overview of our software program. Section 4 of the design document consists of the final object class diagram and all of the attributes, functions, and packages we will code and implement into our game. The most important aspect we focused on is the user experience, we want the users of our game to have a smooth-sailing experience and that any person of any age group can easily navigate and play our game with ease.

**Anibal Ruiz** - API - Liaison:

My main contribution was in majority section 2, the subsystem decomposition. Subsystem decomposition of packages and every single interaction is demonstrated, and then hardware/software mapping and persistent data management issues will be explained. We would like to design the game in a way that the implementation processes would be easier, intuitive and straightforward. We have decided that the Model-View Controller system design pattern is a good fit to apply to our program. We divided our system based on MVC principles: User Interface, Model and Scenes are the subsystems. I also contributed with Hardware/software mapping info, persistent data management and access control and security. The termination and failure are the last two things in this important section.

**Maaz Ansari** - Configuration Manager

My principal contribution for the design document was my work on the High Level Software Architecture and Subsystem Services. Specifically, I worked on the subsystem decomposition, and user interface. I also worked on the scenes and model section of the Subsystem services. Working on the Subsystem Decomposition is important because it allows the system to be decomposed into well defined subsystems. These subsystems are decomposed into well defined components as a result. We need that for our game to run smoothly. Working on User Interface is also integral because it serves as the communication between the human and computer interaction. Without a user interface, the game will not run. Scenes and models are important to visualize how the game will be designed.

**Jean Claude Charles Boute** - Webmaster

My main contribution to the design document was that I was able to continue with adding more material to the subsystem decomposition and the subsystem services. Subsystem decomposition which focuses on the operations, events of the game. Decomposition is dividing the system into smaller systems. Having a game that does not lag and runs at the capacity that is needed is our goal. Part of a subsystem is to be able to manage resources, this game that we are creating we want to make sure that our visuals are easy and understandable with intuitive user navigation. The way our design will be laid out will be familiar to our audience. We want to make sure that the user is engaged in the game.