THE DESIGN AND IMPLEMENTATION OF THE 2D ARCADE GAME DUAL SPACE INVADERS

Victor Montshing (1419577) & Moshekwa Malatji (1387556)

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: This paper documents the design and development of a 2D computer game Duel Space Invaders. The game is developed in C++17 programming language in conjunction with SFML 2.5.0 library for graphics in an Object Orientated paradigm. The doctest framework is used for unit testing to explicitly test five classes which result in 49 successful test cases with 103 assertions. The presented design achieved basic functionality, including five minor and two major feature enhancements. The game design uses the Axis-Aligned Bounding Box(AABB) Algorithm to detect collisions between game objects. An analysis of the overall design commends the strengths and exposes the weaknesses of the design. A notable weakness in the solution is that a class which serves as the game engine is monolithic. One strength is that the design is based on the Layered Architectural Pattern which respects a key programming principle that is separation of concerns. Furthermore, Recommendations which are to strengthen the weaknesses and improve the quality and efficiency of the overall design are documented in the scope of the report.

Key words: Doctest, Object Orientated Design, Separation of Concerns, SFML,Axis-Aligned Bounding Box, Layered Architectural Pattern, DRY Principle, Monolithic

1. INTRODUCTION

Computer programming is about creating abstractions. The abstract objects are implemented in order to realise useful programs which solve problems, improve quality of life and design games to promote creativity and idea generation. This report documents the design, implementation, testing and analysis of the Dual Space Invaders arcade game. Space Invaders is an arcade that was released in 1978 by Taito Corporation and it is considered to be one of the first arcade games created [1]. Dual Space Invaders is a version of Space Invaders which has two players.

The Dual Invaders game offers the user two player objects which are at the top and bottom of the screen and they can both be used in the "single player" & "dual player" game modes. The Single player mode allows the user to control the player object at the bottom of the screen while the other player object mirrors the movement and shooting demeanour of the first player. Furthermore, the dual game playing mode enables the functionality of both game objects to move and shoot independently provided that there are two different users to regulate each game object.

A good way to boost the user's self-confidence and encourage one to continuously play the game is ensuring the user is always engaged with the game, this is achieved by preserving the high score of every game such that in the next game, the user can try to exceed the current high score. It has to be noted that the two game playing objects have a common score which progressively adds to the current high score. The game ending state is triggered when one of the following event occurs: all the aliens are destroyed by the player object, either of the player laser cannons are destroyed by shooting each other or the aliens shots and when the aliens seize the player's territory i.e. entering the top or bottom row. The scope of the report provides an overview of the Object-Oriented design methodology that was involved in the design and implementation of the Dual Space Invader game. The presentation, data and logic layers of the overall design are discussed. The unit testing of the game objects is an imperative part of the design. Object-Oriented design is also discussed along with the analysis of the derived functionality and further improvements to be implemented in order to achieve a better functional and efficient design.

1.1 Assumptions & Constraints

In order to derive an acceptable design as stated by [1], the following constraints to be adhered to are as follows:

- The Dual Space Invader game is programmed in ANSI/ISO C++ and the SFML 2.5.0 library is to be utilized for graphics and textual representation provided that no earlier versions of SFML are not used.
- It is not allowed to use external APIs such as OPENGL, Unreal and Ogre3D e.t.c
- Libraries and other frameworks built on top of SFML may not be used
- It is imperative that the game executable can run on the Microsoft Windows Operating System
- The game display screen resolution should not exceed 1920 x 1080 pixels
- The Unit Testing framework to be used is to be used to provide unit tests in order to validate game functionality

1.2 Success Criteria

The design and implementation of the game is to be deemed successful provided that it satisfies the following criteria:

- The logic and presentation layers of the game are to be properly separated
- Basic Functionality, two minor and major feature enhancements provided by [1] are to be implemented.
- All aspects of the game functionality and logic such as collisions are to be meticulously tested
- Upon implementation of the game, good object-oriented design, practices and principles are to be used to develop an efficient and functional game design

2. SOFTWARE DESIGN

2.1 Software Architecture

The design of the game architecture is implemented based on the concept of *Separation of Concerns*, this concept makes it convenient to develop, maintain and test different functions of the design. This concept is a major feature of the Layered Architectural Pattern which also serves as the building block of the game software architecture. The benefits of using the Layered Architectural Pattern[2] is that it allows for isolation of layers in order to make it convenient to maintain and test each layer. The isolation of layers ensures that there isn't a cause-effect i.e. changing one layer which ultimately affects other layers, this approach ensures that the presentation layer doesn't have knowledge of the logic layer, as the logic layer has limited or small knowledge of the presentation layer. *Figure 1* denotes a high level representation of interactions between game layers.

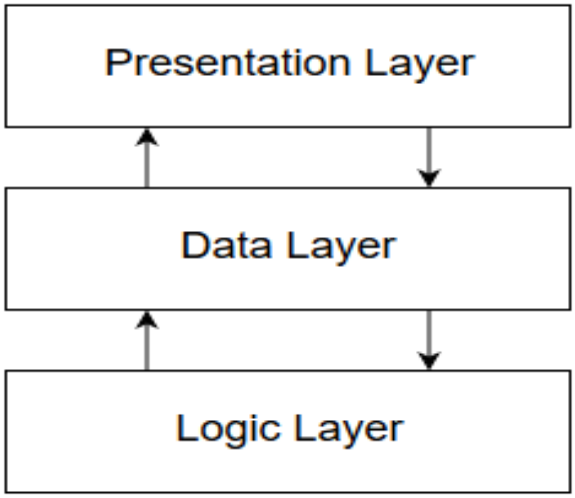

Figure 1 : The Layered Architectural Pattern Design Flow-diagram

The game design has the presentation layer, data layer and the logic layer. The presentation layer serves as a high level layer which accepts input from the user and the data/instructions are stored in the data layer before it enters the logic layer. The logic layer is responsible for decision making as it serves as a fuel to the game engine which interprets the instructions/data from the user and produces a valid output.

2.2 Object-Oriented Design

A key part of the Object-Oriented Design of the Dual Space Invaders game is to use concept of Inheritance hierarchy. The inheritance hierarchy ensures that a child class inherits methods and properties of its parent class, this key technique plays a vital role in the design as it accommodates the use of *Polymorphism* in different classes. Another advantage of using the inheritance hierarchy is that the code can be re-used and overall space and time complexity of the design is reduced.

3. THE PRESENTATION LAYER

The presentation layer serves the first part of the Layered Architecture Pattern of the Dual Space Invader Game design. This layer makes use of the SFML 2.5.0 library in to capture user inputs from the keyboard, draw various game objects and screen displays for different game states. The isolation of the presentation layer from the data and logic layer makes it feasible to replace SFML 2.5.0 and use another framework or library to handle the presentation functionality.

3.1 KeyHandler

The KeyHandler class is responsible for accepting user input from the keyboard. This class contains the KeyHandler() and KeyHandler2() member functions which evaluate user input in order to regulate the LaserCanon and LaserCanon2 game objects, respectively.

3.2 ImageDrawer

The ImageDrawer class provides the game with the ability to draw game sprites viz. Aliens, Lasers, LaserCanon and LaserCanonShield. This class has member functions which accept an object of the game object by reference as a parameter that is to be drawn. ImageDrawer class also draws the home, game won and game lost screens.

3.3 ImageLoader

ImageLoader class' main purpose is solely load all game images to be associated with game objects. This class consists of a vector of shared points of type Sprite which is responsible of creating Sprite objects.

3.4 *WindowDisplay*

Upon creation of WINDOWDISPLAY class it was important to adhere to the constraints(Section 1.1) of the design which restricts the maximum screen resolution to be 1920 x 1080 pixels. The WINDOWDISPLAY class has a shared pointer to a SFML Render Window and the size of the window is 400 x 500 pixels at 60 Frames Per Second(FPS) which satisfies the screen resolution restriction. Furthermore, The Render window is passed by this class to the IMAGEDRAWER class in order for game objects to be drawn to the current game window. This allows the ImageDrawer to be able to draw the Game Won and Game Lost screens on the render window. The WINDOWDISPLAY class is responsible to handle user input for setting desired game mode on the splash screen window as listed on the menu:

- Single Player Mode(where the game mode is set by pressing "L").
- Dual Player Mode(where the game mode is set by pressing "K").
- Instructions for Player one and Player two on how to move and shoot the first and second LASERCANON object, respectively.
- Exit state(where the game mode is set by pressing "ESC" and terminates the program execution).

4. THE DATA LAYER

The data layer serves as the second phase of the Layered Architectural Pattern of the Dual Space Invader Game design. This layer's main purpose is to store the information of the various game objects and it provides information to the logic layer to be processed, aggregated and interpreted. The presentation layer also receives information such as coordinates and life state of the objects from the data layer in order to draw the sprites.

4.1 *GameEntity*

The GAMEENTITY class is the backbone of the data layer architecture. This class plays a vital role in the creation of game objects as it initialises the game object's coordinates and ensures that they are valid i.e. the coordinates are within the boundary of the Render Window. Invalid coordinates are handled by INVALIDENTITYPOSITIONS exception. The Game Entity ensures that every created game object has a boolean variable *_life = true* and the member function *giveEntityLife()* is called to realize the life state of the game object, whilst the *destroyEntity()* member function destroys an object's life by ensuring that the boolean *_life = false*. The game is implemented such that the LASERCANON object has a limited lifespan, therefore the GAMEENTITY class initialises and returns the number of lives for every created LASERCANON object. The GAMEENTITY class is a parent class to other classes of the data layer which inherit its properties and methodology, refer to *Figure 2* for the inheritance hierarchy of the GAME ENTITY class.

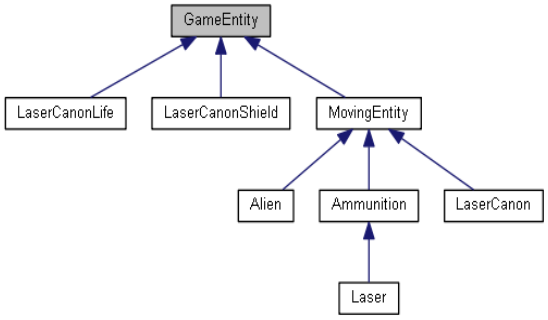

Figure 2 : Inheritance diagram of Game Entity

4.2 *MovingEntity*

This class is a child class of the GAMEENTITY class and it inherits various properties which aid in fulfilling its primary objective that is to regulate the movement of the game objects. The constructor of this class initialises the speed of the player and the GETENTITYSPEED() member function returns the speed of the LASERCANON object.

4.3 *Ammunition*

The AMMUNITION class is a child class of the GAMEENTITY class and it inherits the capability to set *_life = true* when a game object shoots a laser, this is achieved by calling the member function *giveEntityLife()*.

4.4 *LaserCanon*

As seen in *Figure 2*, the LASERCANON class is derived from the MOVINGENTITY class in order to allow a player to move in the player's desired direction. This class is also responsible for continuously updating the score and high score of the player and the game, respectively. Furthermore, this class throws FILECANNOTBEOPENED exception if a text file which contains the score and high score cannot be opened. It has to be noted, that the INVALIDLASERCANONCOORDINATES exception is thrown provided that a LASERCANON object is to be created at coordinates which are not recognised by the design of the game.

4.5 *LaserCanonLife*

This class is responsible for keeping data on the number of lives a player has during the gameplay. Hence, the LASERCANONLIFE class is a child class of the GAMEENTITY class. The Laser Canon has a total

lifespan of 3 and a player life decreases when an Alien shoots the player and if the two players collide or when they shoot at each other. This class This subject of collision handling and detection is further elaborated on Section 5.3.

4.6 Laser

Although Laser Canons are able to shoot lasers, it is important that game objects such as the Aliens and are capable of shooting lasers in order to make the game more challenging. The LASER class ensures that LASER objects of the ALIENS or the LASERCANON objects are able to move vertically to the direction the object is facing. This class is a child class of the MOVINGENTITY class as it is imperative that it inherits the movement functionality. Creating a LASER object for an ALIEN class object requires that the created object has the same coordinates as the alien, therefore, this class throws a INVALIDALIENLASERCOORDINATES exception if this condition is violated. Similarly, a INVALIDLASERCANONCOORDINATES exception is executed should a LASER object of the LASERCANON be initially created outside of the bottom or top boundaries of screen which are specified initial positions of the laser canons.

4.7 Alien

There exists a "has-a" relationship between the LASER class and ALIEN class, which means that destroying an alien also destroys the laser associated with the specific alien. The game consists of two sets of Aliens; the first set of three rows of aliens move upwards while the second set moves downwards. The ALIEN class ensures that each set of aliens have the movement functionality, hence it is a subclass of the MOVINGENTITY class. The class ensures that movement of the aliens is restricted to the boundaries of the screen, provided that the alien object is initialized with coordinates that are not out of the scope of the game window therefore an INVALIDALIENCOORDINATES exception is responsible to avoid violation of this condition. An enumeration ALIENCOLOUR provides colours for red, purple and green aliens.

4.8 LaserCanonShields

The game offers security to the laser canons by creating LASERCANONSHIELDS objects which can handle the impact of the laser shots from the alien. However, player shield objects of the LASERCANONSHIELDS are destroyed once they reach an unstable state. This class is derived from the GAMEENTITY class and inherits the properties of the parent class by setting the life state of every LASERCANONSHIELDS object. A shield's life state *_life = true* upon creation and the member function *giveEntityLife()* is called to realize the life state of the game object, whilst the *destroyEntity()* member function destroys an object's life by ensuring that the boolean *_life = false* when the shield reaches an unstable state prior to destruction.

5. THE LOGIC LAYER

The logic layer is the game engine which fuels the logic of the Dual Space Invader game. This layer manages the demeanour, interaction and the overall conversations between game objects. The Layered Architecture Pattern as indicated in *Figure 1* ensures that the logic layer gathers information from the data layer in order to process and make decisions which produce a logical output.

5.1 GameLoop

The GAMELOOP class serves as the skeleton which the game engine is based on. This class is responsible for the creation of all game objects, maintaining game activities and the carrying out overall functionality of the game. In essence, the GAMELOOP class is the interaction of the presentation and data layer in the Layered Architectural Pattern design. *Figure 4* in the Appendix indicates the Sequence of interactions between objects of GAMELOOP class. The GAMELOOP class allows for conversations between objects during gameplay which includes processing data from the data layer in order to process and make decisions which produce a logical output to the presentation layer.

5.2 GameUpdater

Another important aspect of the game logic is the ability to realize the decisions which are made by objects as a result of different user inputs and game activities. The GAMEUPDATER class constantly updates the game as a consequence of events which occur during gameplay, therefore upon movement and collisions of the game objects this class updates the coordinates.

5.3 CollisionDetector

The COLLISIONDETECTOR class' primary purpose is to handle the collision of game sprites/objects. It is imperative that collisions are detected and handled as this denotes if a player has won or lost the game or if a player has lost a life. This class caters for collisions between the following game objects:

- Laser Canons: A player loses a life if two Laser Canons collide. The game ends and the player loses if there are no lives remaining.
- Laser Canon and Laser: A collision occurs if an Alien shoots the Laser Canon or if the Laser Canons shoot each other. In both instances, the player loses a life and if there are no lives remaining the game ends in a loss.
- Laser Canon and Laser Canon Shield: A Laser Canon object cannot move through the Laser Canon Shield if and only if the Laser Canon

Shield object is not destroyed.

- Laser Canon Shield and Laser: Alien objects continuously shoot at the Laser Canons and the shields provide protection. Collisions between Laser Canon shields and Lasers are handled by this class provided that the shield is still present and has not been utterly destroyed by the lasers
- Alien and Laser: In order for the player to win the game, all alien objects are to be destroyed by the Laser Canons regardless of the game playing mode. Aliens are destroyed by the Laser shot of the Laser Canon objects upon instruction of the user.

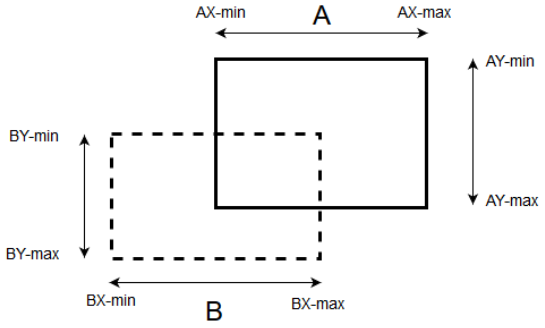

Figure 3 : Axis-Aligned Bounding Box diagram

Figure 5 in the Appendix is a flow-diagram which denotes the logic of CollisionDetector class. This class uses the Axis-Aligned Bounding Box(AABB) Collision Detection Algorithm, which detects a collision between game objects when there is an overlap of the rectangular regions of the two game objects/sprites. The AABB algorithm is preferred over the Circle Collision Algorithm as game sprites are rectangular shapes. *Figure 3* illustrates the collision of two game sprites A and B with regular coordinates $(A_{x-min}, A_{y-min}, A_{x-max}, A_{y-max})$ and $(B_{x-min}, B_{y-min}, B_{x-max}, B_{y-max})$, respectively. The AABB algorithm detects a collision between the two sprites if and only if $(B_{x-max} > A_{x-min} \&\& B_{x-min} < A_{x-max})$ and $(B_{y-min} > A_{y-min} \&\& B_{y-max} < A_{y-max})$.

6. GAME TESTS

It is imperative to perform unit testing in order to improve the quality of the design and identify logical errors in the source code. The DOCTEST framework is used to perform the unit testing as it makes it convenient to validate that the game functions as it was designed.

6.1 Initialisation & Position

The game entities are ensured to be positioned at the valid initial positions within the game screen. Initially, Laser Canon and the Laser are to be positioned at the bottom and top of the screen while the Aliens are to be positioned in the middle of the screen. This part of the unit test also validates the SET and GET functions of the various game entities.

6.2 Exception handling

It is necessary to code defensively and handle invalid inputs in order to produce an efficient design which doesn't produce errors in run time. This unit testing aspect focuses on throwing exceptions to cases which can cause the game to crash. Tests for the exception handled include to inhibit the creation of game objects which have been destroyed and trying to initialise a game object outside the boundaries of the screen.

6.3 Shooting

The Laser Canon and Alien's shooting capabilities are tested and deemed successful. The unit tests ensure that both game objects(Laser Canon and Aliens) are able to shoot lasers when a specified shooting key is pressed.

6.4 Collision Handling

The collisions between the game objects (the laser, laser canon, laser canon shields, Alien) are tested to ensure they are functional as desired. It is imperative to ensure the collision detection is functional as the game ending states mentioned in Section 1 are solely dependent on this aspect.

6.5 Movement

The game objects which are movable are tested to ensure that they function as desired by the logic layer. The movement of the laser canon is tested to validate that it responds to move in the desired direction of the key that is pressed by the user. Furthermore, game objects which move independent of a key pressed by the user such as the Aliens are tested to ensure they move in the correct direction as per the logic layer.

Class	Tests Cases	Assertions
Game Entity	2	5
Laser Canon	15	35
Laser	8	15
Laser Canon Shield	5	11
Alien	11	21
Alien Laser	8	16

Table 1 : Summary of Unit tests for Game Classes

It has to be noted that although unit testing improves the quality of the design it does not imply that other key programming principles are to be ignored in the design. The logic layer of the game design is such that the code is readable by using appropriate control flow statements, consistent indentation and making class-functions read-only where necessary. *Table 1* is a summary of the unit tests which includes the number of tests and tests of the game classes which were successfully tested.

7. ANALYSIS OF THE DESIGN

In section 1.2, it is mentioned that according to [1] the minimum requirements overall design has to consist of basic functionality. The documented design consists of additional five minor and two major features which are successfully implemented.

7.1 Minor Feature Enhancements

- Aliens are capable of shooting at the Laser Canons and get destroyed if shot by the Laser Canons
- The game graphics are good and alien animation is implemented (i.e they appear to be flying as the move down or up the screen)
- The Laser Canons have shields which protect them from the Alien missiles, these shields are progressively destroyed by the missiles
- There exists a scoring system and the ability to save and display the high score from one game to the next game
- The player has a limited life span which is more than one and the lifespan is indicated by four sprites on the top right corner of the screen which are reduced as the player loses a life

7.2 Major Feature Enhancements

- The game offers a single-player mode whereby a player can only regulate one Laser Canon and the other Laser Canon automatically moves in the opposite direction of the regulated Laser Canon
- A player is able to move a Laser Canon (upwards or downwards) and join the other player on the opposite side of the screen. During the vertical movement of the Laser Canon: collisions with the aliens destroys a player, the player cannot move horizontally until it reaches the opposite side of the screen and the player has firepower to shoot in the direction of propagation. The Laser Canon is able to face the direction of the other Laser Canon and fire shots.

7.3 Quality of Design

The overall design of the Dual Space Invaders game has some trade-offs which include notable strengths and weaknesses.

7.3.1 Strengths:

- The usage of Layered Architectural Pattern Design which is separating the code into three layers viz the presentation, data and logic layers to ensure that code is easy to maintain, debug, test and allows for flexibility in a choice of frameworks
- Implementation makes use of the inheritance hierarchy to reduce time and space complexity as code is reused and the code structure is short and well defined
- The Implementation also caters for information hiding as different classes protect their private member variables

7.3.2 Weaknesses:

- The complexity of GameLoop class is quite large as it has a lot of responsibility
- During gameplay, upon detecting collisions game objects are not instantly deleted i.e. memory is not freed. The objects are only literally deleted when program execution is terminated
- In the EntityShape class there exist a violation of the DRY principle. Member functions of this class all accept a game object and return the dimensions of each object, this is an instance of code repetition and measures to counter this are mention in the succeeding section

8. RECOMMENDATIONS FOR FURTHER IMPROVEMENTS

Section 7.3.2 includes weaknesses which arise from the design of the Dual Space Invader Game which can be improved to develop a much more robust and efficient design. The complexity of the GameLoop class can be further reduced by allocating certain responsibilities of the class to other classes such as the creation of objects and executing game activities. In order to avoid the violation of the DRY principle regarding the EntityShape class, a generic TemplateFunction[3] which accepts a game object and returns the dimensions of the game object can be implemented. The generic function replaces all the functions which accept different objects but return same parameters of the objects. It is also recommended that further implementation should include collision detection and handling for sprites which are not rectangular in shape. The Circle Collision Detection for circular shaped sprites is to serve as an algorithm for this purpose.

9. CONCLUSION

The design of the Dual Space Invader game based on the Layered Architectural Design is successfully implemented. The overall design successfully exceeds the minimum requirements as it satisfies basic functionality which includes five minor and two major feature enhancements. Furthermore, the design and implementation is such that the key programming principles which are the DRY principle, separation of concerns and code re-usability. Unit testing is performed in order to identify errors validate the functionality of the source code. An analysis of the overall design commends the strengths and exposes the weaknesses of the quality of the design, however, further improvements are to be made to the overall design in order to derive a robust and efficient design to improve overall functionality and quality of the design.

REFERENCES

[1] S. P. Levitt. "Project 2019 - Duel Invaders." *Electrical and Information Engineering, Software Development 2*, p. 1, 2019.

[2] C. D. H. M. A.B. Belle1, G. El Boussaidi. "The Layered Architecture revisited: Is it an Optimization Problem." In *The 25th International Conference on Software Engineering and Knowledge Engineering*. Montreal,Canada, 2013.

[3] R. T. M.T. Goodrich and D. Mount. *Data Structures and Algorithms*, chap. 2, pp. 90–91. John Wiley Sons Inc., second ed., 2011.

Appendix

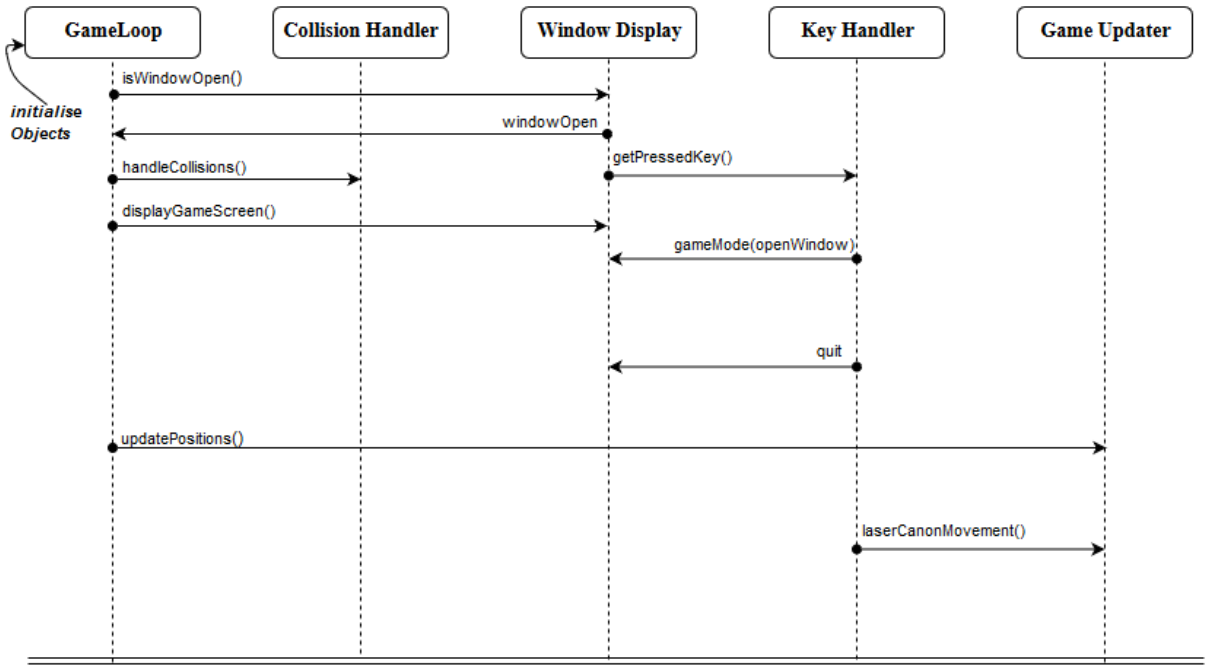

Figure 4 : Sequence diagram of interaction between classes

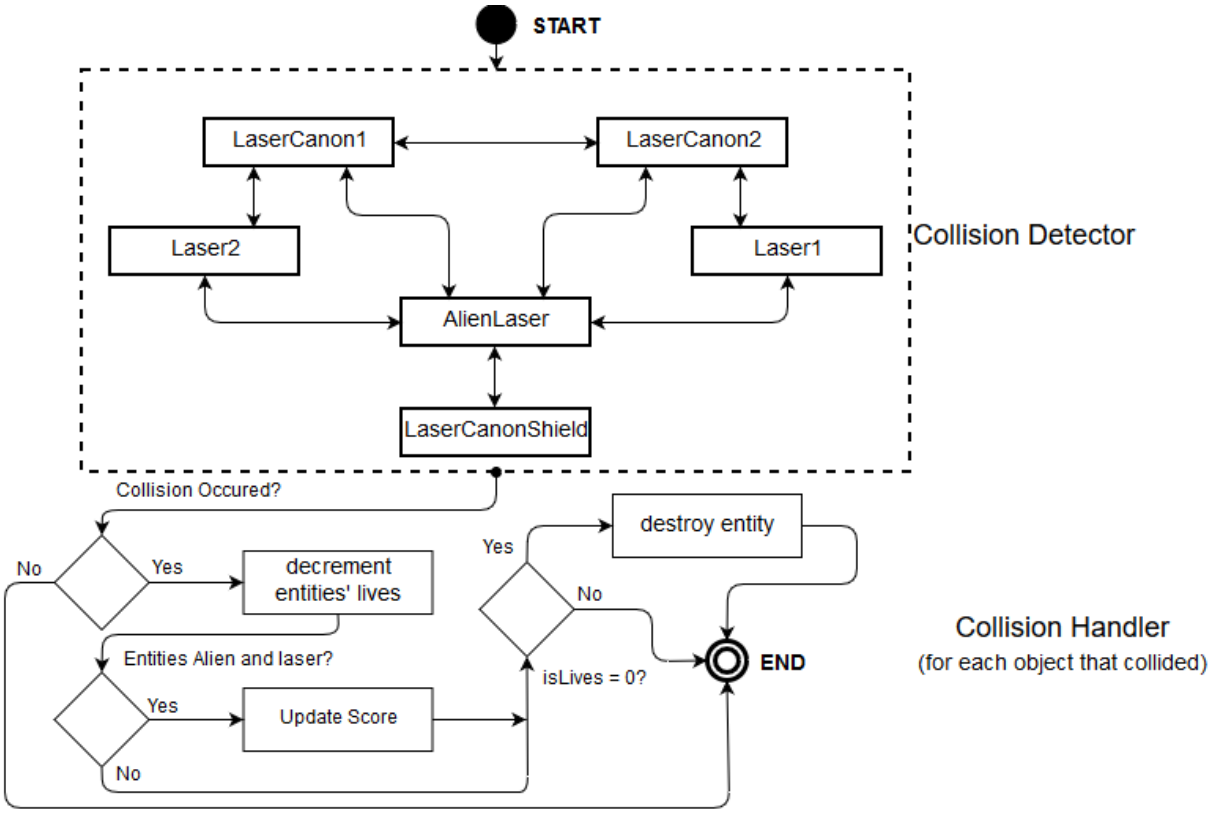

Figure 5 : Collision detector and handler logic diagram