**BUCHAREST UNIVERSITY OF ECONOMIC STUDIES**

FACULTY OF ECONOMIC CYBERNETICS, STATISTICS AND INFORMATICS

A close up of a logo

Description automatically generated

**Bachelor Thesis**

**Video Game Design and Development Using Unity and Web Technologies**

**Supervisor**

Prof. **Liviu-Adrian COTFAS**, PhD.

**Student**

**Vrînceanu Alin-Vlăduț**

Bucharest

2023

Declaration regarding the originality of the content and the assumption of responsibility

I hereby declare that the results presented in this thesis are entirely the result of my creation, unless references are made to the results of other authors. I confirm that any material used from other sources (magazines, books, articles and Internet sites) is clearly referenced in the work and is indicated in the list of bibliographic references.

# Table of Contents

[Table of Contents 3](#_Toc139041325)

[1. Introduction 4](#_Toc139041326)

[2. Technologies Used 6](#_Toc139041327)

[2.1. ASP.NET Core MVC 6](#_Toc139041328)

[2.2. Microsoft SQL Server Database 11](#_Toc139041329)

[2.3. Unity Game Engine & WebGL 14](#_Toc139041330)

[2.4. Game Core Algorithms 18](#_Toc139041331)

[3. Web Application & Game Design Approach 24](#_Toc139041332)

[3.1. Web Application Flows 24](#_Toc139041333)

[3.2. Web Application Views/Pages 26](#_Toc139041334)

[3.3. Game Design Brief 32](#_Toc139041335)

# Introduction

Over the past decades, video game development has faced remarkable events, trends, and technological breakthroughs that led to significant changes in how a game is designed and developed, thus evolving from rudimentary mechanics, graphics, and physics to immersive experiences akin to living in an alternative reality. Thanks to the improvements in the hardware capabilities of the newly produced devices, advancements in development software and engines, and, significantly, broad access to high-speed internet connectivity, video games are now in a better position than ever.

This thesis explores the importance of video game design and development using Unity and Web technologies, emphasizing the integration between the 2D Space Shooter game "Stellar Strike!", which presents advanced algorithms for handling game objects' behavior, and the ASP.NET Core MVC framework, renowned for its rich documentation for building web apps and the implementation of the Model-View-Controller design pattern.[1]

The relevance of this theme cannot be overstated, as the global gaming market has reached $202.64 billion in 2021, displaying a significant compound annual growth rate of 10.2% until 2030.[2] Exploring the technologies that pose a critical role in shaping the gaming industry is crucial for understanding its success across the years. Considering the demise of Adobe Flash Player, which was the principal standard for delivering rich Web content, including games, until December 2020, new alternatives were considered. Given the ease of access to browsers and their flexibility, the chances of browser-based games ending once with Adobe Flash Player were slim to none. That is how Unity, with its WebGL build capabilities, has emerged as a major player in Web video games.

Since Adobe Flash Player was so widely used across the internet, combined with the fact that it was showing signs of poor security, it became a target for hackers with malicious intent. Other factors, such as the popularity of smartphones, which have had a hard time supporting Flash Player, and the high-performance resources required led to the decline and subsequent discontinuation of Adobe Flash Player in 2020, thus leading to a large number of games and applications becoming obsolete. [3] Among the viable alternatives, Unity's WebGL has surfaced as a potential contender, offering the benefits of advanced game development capabilities and the ability to run games directly in the web browser without needing plugins.

This transition is significant in the case of "Stellar Strike!" a 2D space shooter application developed with Unity and hosted via WebGL on an ASP.NET Core MVC website. This game is not just proof of how advanced browser games can be created today; it also reaffirms the potential and flexibility of Unity and WebGL. Several similar applications have been created, leveraging the power of Unity. A variate library of such games can be found on websites like "Y8", which contain examples of Unity WebGL games that are pushing the boundaries of browser-based gaming. The uniqueness of "Stellar Strike!", however, lies in its own algorithms used for handling the game objects combined with their deployment on an ASP.NET Core MVC website that displays a seamless integration between gaming development and web technologies.

There are several advantages to Unity's WebGL over the alternatives currently available. It offers a more powerful graphic capability, allowing Cross-Platform Development and eliminating all other plugins that must be installed to run a game in the browser. In addition, it provides a safer environment for users thanks to its more secure and updated framework compared with Adobe Flash Player. Flash's death marked a turning point in the history of web technologies and browser-based gaming. However, a new era has begun, bringing the legacy of browser gaming to life but also expanding upon it with tools like Unity WebGL that help game developers create more immersive, interactive, and engaging games.

The importance and actuality of developing web-based video games using modern technologies like Unity and WebGL are multi-dimensional. An increasing volume of activities, such as gaming, are moving to the internet platform in view of the development of the Digital Age. This trend was rapidly accelerated by the COVID-19 pandemic, which started in 2020 and has continued to disrupt world behavior, resulting in a significant increase in demand for electronic entertainment, such as online games. The leading technologies in this new generation of Web-based game development are Unity and WebGL. It enables developers to create complex, highly detailed games that can be played directly within a Web browser and reduces the difficulty for users in accessing such games while allowing them to reach an increased audience. This immediacy of access represents a shift in how games are distributed and consumed, aligning with the increasing demand for instantaneous, on-demand digital content.

The continuing evolution of hardware and internet capabilities further amplifies the actuality of this topic. Games reserved exclusively for dedicated gaming consoles or high-end desktop computers are now being played directly in the browser, with more powerful devices and faster internet speeds becoming common. The ability to deliver these experiences to a global audience over the web represents a considerable shift in the gaming industry's landscape. The Adobe Flash Player discontinuation and subsequent proliferation of tools such as Unity and WebGL represent an excellent example of how the technology landscape is constantly evolving. Being aware of these changes, not only adapting to the new situation but also seizing opportunities that arise from them, is essential for software developers and technologists.

# Technologies Used

## ASP.NET Core MVC

The foundation of the "Stellar Strike!" website's infrastructure is ASP.NET Core MVC, a high-performing and freely available framework. The decision to use this technology was driven by its capacity to create robust, secure, and scalable web applications that can efficiently serve high-quality WebGL games like "Stellar Strike!" to end-users. The MVC (Model-View-Controller) architectural pattern, intrinsic to ASP.NET Core, was instrumental in effectively managing different aspects of the "Stellar Strike!" web application. The MVC architecture enables a structured and practical development approach by decoupling an application from three interdependent components, which are Models, Views, or Controllers.

The project has integrated user authentication and safety management through the core identity of ASP.NET. The library offered a solution to user registration and login that would provide an efficient, safe user experience out-of-the-box. Users could easily log on to the website and play the game once they had successfully registered.

Several functionalities associated with ASP.NET Core Identity were integrated to enhance the user’s experience and security:

1. **Data Download and Deletion:** Users were given control over their data, reflecting the principles of data protection regulations like the EU's General Data Protection Regulation (GDPR). Per the principle of "Right to Be Forgotten" and "Right to Access," users can download their data or request its deletion. (Figure 1)

A screen shot of a computer

Description automatically generated with medium confidence

Figure 1 - DeletePersonalDataModel class

A screen shot of a computer program

Description automatically generated with low confidence

Figure 2 - Handling User Deletion

Three services are injected in the constructor: UserManager, SignInManager, and ILogger. These are standard services for handling user operations and logging in ASP.NET Core.[4]

An InputModel class is defined with a single property Password. This class is intended to bind to the form data submitted from the view (webpage). (Figure 3)

OnGet method: It's called when the page is initially requested. It checks if the current user exists, and if not, it returns a NotFound result. If the user does exist, it checks if the user has a password and sets the RequirePassword property accordingly. (Figure 2)

OnPostAsync method: It's called when the form on the page is submitted. It performs the same user existence check as OnGet. If the user has a password, it verifies the password is correct. If the password is incorrect, it adds an error to the ModelState and returns the page, so the error can be shown. If the password is correct or not required, it attempts to delete the user. If the deletion is successful, it signs out the user, logs a message indicating that the user deleted themselves, and redirects to the home page.

1. **Account Verification:** To prevent the creation of fraudulent accounts, a user verification system was implemented. Users were asked to verify their accounts to ensure the validity and ownership of the email sent during a registration session.

A picture containing text, screenshot

Description automatically generated

Figure 3 - InputModel for Registration

For the account verification, there has also been set up a smtp server which handles the messaging. Papercut, a lightweight open-source software that emulates the SMTP Simple Mail Transfer Protocol server, was used to build this server. It is primarily used in applications for the local development and testing of email functionality. (Figure 5)

A picture containing text, font, screenshot

Description automatically generated

Figure 4 - Email Settings



Figure 5 - MailKit Configuration in Program.cs

The SMTP server information to send verification or password reset email can be found in the configuration appsettings.json used along with other settings such as connection strings, used for connecting to the Microsoft SQL Server database, and logging settings. (Figure 4)

Get and Post Methods: The OnGetAsync method is executed when the page is requested via HTTP GET, and it initializes necessary data such as ReturnUrl and ExternalLogins. The OnPostAsync method is executed when the registration form is submitted via HTTP POST. It validates the input data, checks for existing users with the same email or username, creates a new IdentityUser, saves it to the database using UserManager, generates an email confirmation token, sends a confirmation email using \_emailService, and handles the registration flow based on the RequireConfirmedAccount option. (Figure 6 )

The code includes a couple of helper methods, CreateUser and GetEmailStore, which are used internally to instantiate IdentityUser and retrieve the email store respectively.

A screen shot of a computer program

Description automatically generated with medium confidence

Figure 6 - Handling the Registration and Email sending

1. **Password Reset:** Users could easily change their password in the event that they forgot it. By providing a quick solution to common problems encountered by users, this feature improved the user experience.

A similar implementation of the email verification from registration methods can be found in password reset. In both cases, the objective is to send an email that contains a unique identifier or link that allows users to perform specific actions, in this case, to redirect the user to the confirmation pages for verifying the email address and changing the password, respectively. (Figure 7)

A screen shot of a computer code

Description automatically generated with low confidence

Figure 7 - Handling Password Reset

1. **Password and Email Change:** Users were given the option to change their password and email. In addition to providing flexibility for users, this functionality also provides an additional layer of security that allows the user to change his or her password at any time.

A picture containing text, screenshot, software

Description automatically generated

Figure 8 - Handling Password Change

The code will first check whether the model state is valid, and if not, it will return the current page with validation errors. It will attempt to find the currently logged-in user and return a "Not Available" message if it can't find him. The code will set a new password for the user, assuming that the Model State is intact and you have managed to recover the user. When adding the password, it refreshes your user's SignIn information and sends a success message if you succeed. (Figure 8 )

The integration of the Unity WebGL build with the ASP.NET Core MVC framework was seamless, providing an excellent environment to host "Stellar Strike!". The framework's scalability and robustness ensured that the game could be served to a multitude of users simultaneously, offering a smooth gaming experience regardless of the load on the server.

## Microsoft SQL Server Database

For "Stellar Strike!", the data persistence layer was managed by the Microsoft SQL Server 2019, a leading relational database management system. The choice of this database system was made in view of its advanced authentication features, exceptional performance, and excellent compatibility with the ASP.NET Core MVC framework. (Figure 9)

The database was structured with two essential tables: "AspNetUsers" and "BoardItem". (Figure 10)

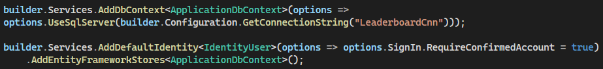


Figure 9 - Database Configuration in Program.cs

The ASP.NET Core Identity framework allows the management and retention of user accounts in ASP.NET Core applications. If an individual user account has been selected as the authentication mechanism, Identity will be added to the project. By default, Identity makes use of an Entity Framework (EF) Core data model. [5]

A screenshot of a computer

Description automatically generated

Figure 10 - Database Diagram

"AspNetUsers," table was included in ASP.NET Core Identity Framework and migrated to the database. Critical data relating to users, for example, user names, passwords, and email addresses were stored in this table. For the purpose of managing user authentication tasks, this table's data is used by registration and login controllers. The use of this table with ASP.NET Core Identity ensures that user data is handled in a secure and efficient manner. (Figure 11)

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure 11 - IdentityUser Model

The JSON configuration file appsettings.json holds configuration settings, such as strings, for the connection to the database. The following entry has been added to describe the connection with the database which will be used for Stellar Strike data: (Figure 12)

Figure 12 - Database Connection String

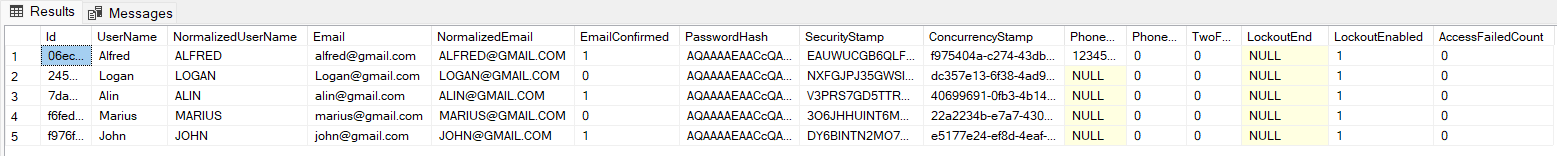


Figure 13 - AspNetUsers Entries

In (Figure 13) it is presented an example of how user data is organized within the 'AspNetUsers' table in a typical ASP.NET application. This table is the standard storage location for user account information when using ASP.NET Identity for user management. Passwords are not stored in plain text. This is due to the process of password hashing, a crucial technique in today's information security, PBKDF2 being the specific password hashing algorithm used for this purpose.[6]

The second table, "BoardItem," was created to manage the game-specific data—specifically, the scores achieved by the users in "Stellar Strike!". Each record in the "boardItem" table is a unique play session; its score was stored for it. A foreign key relationship with the "AspNetUsers" table using the user's username, which is unique for each user, has been created to link this data to the corresponding user. This setup allowed for accurate tracking of each user's scores, adding a competitive edge to the game and enhancing the overall user experience. (Figure 14)

A screen shot of a computer

Description automatically generated with medium confidence

Figure 14 - LeaderboardItem Model

**ApplicationDbContext** class is derived from the IdentityDbContext class, which is a class provided by ASP.NET Core Identity and is an implementation of DbContext for Identity. IdentityDbContext includes DbSet properties for User, Role, and other Identity-related entities, and ApplicationDbContext extends this context with application-specific entities. (Figure 15)

A screen shot of a computer program

Description automatically generated with low confidence

Figure 15 - Application Database Context

Usage of Microsoft SQL Server 2019 ensured explicit data management for the "Stellar Strike!" web application. A smooth operation of both user and game functionalities was facilitated by seamless integration with ASP.NET Core MVC, together with the efficient structure of database tables. The successful implementation of Microsoft SQL Server 2019 as a data management tool for Web Based Games demonstrates the capability to handle these tasks.

## Unity Game Engine & WebGL

The core of "Stellar Strike!" was built using the Unity game engine, a powerful tool widely used in the gaming industry for its comprehensive features, cross-platform compatibility, and user-friendly interface. It has been an excellent choice for this project because of Unity's flexibility, as well as its support for 2D game development. The game was designed around three 2D scenes: the Main Menu, the Gameplay scene, and the Game Over scene. In structuring and managing these various phases of the game, Unity's multiscene editing tool was essential. All these scenes contained multiple objects, instantiated from prefabs (pre-configured game object templates), that held various scripts to control their behaviors. The use of prefabs allowed code reuse and consistency throughout the game, since any changes to a prefab were reflected in all instances of it.

To ensure compatibility with modern web browsers and to allow the game to be hosted on the ASP.NET Core MVC website, the Unity project was designed to build for WebGL. The WebGL construction option for Unity makes it possible to compile C# script files in JavaScript at a lesser level so that the game runs without installing any plugins.[7]

A screenshot of a computer

Description automatically generated with medium confidence

Figure 16 - Unity Build Settings

The build settings used to configure the Unity WebGL project are shown in Figure 16 - Unity Build Settings. With the "runtime speed" option selected in the Code Optimization field, the settings have been optimized for performance. If this option is selected, it will prioritize the code's performance at runtime to ensure a smooth gaming experience in the browser.[8] (Figure 16)

Unity generated a folder of files, including an HTML shell, JavaScript, and data file after the project was built. These files are a compiled game that can be run in the web browser. These files have been added to the root folder of the ASP.NET Core MVC Web site to integrate the "Stellar Strike!" game into the Web application.

An essential component of this integration is the communication between the Unity game and the ASP.NET Core MVC website. To handle this communication, a custom C# script has been created in Unity, named "ScoreSender.cs" That script started as soon as the game player died, sending Unity Web requests to the web server. This request contains the player's score, which was then processed on the server side. (Figure 17)

A picture containing text, screenshot, software

Description automatically generated

Figure 17 - Game's SendScore.cs

Briefly, the script contains a ScoreSender class which has the following principal methods:

1. **SendScore Method:**
   * The "SendScore" method is responsible for sending the player's score to the server.
   * It logs the final score using Debug.Log.
   * It starts the "SendScoreCoroutine" coroutine to handle the score transmission.
2. **SendScoreCoroutine Coroutine:**
   * The "SendScoreCoroutine" coroutine handles the transmission of the score to the server using HTTP POST method.
   * It defines a URL for the server endpoint where the score will be sent.
   * It creates a new "LeaderboardItem" object with a hardcoded username (currently set as "John") and the player's score.
   * The "LeaderboardItem" object is converted to a JSON string using JsonUtility.ToJson.
   * It creates a UnityWebRequest object with the specified URL and request method set to "POST".
   * The JSON string is converted to a byte array and assigned as the uploadHandler for the web request.
   * A DownloadHandlerBuffer is attached to the web request to receive the response from the server.
   * The web request is sent using "yield return www.SendWebRequest()", which waits for the request to complete.
   * After the request completes, it checks if there was a connection error and logs the appropriate message using Debug.Log.

It is considered the best practice when building games with Unity to contact a database in an indirect way by using web servers instead of direct connections from the game client. Factors related to security are primarily the reason, but there are practical considerations:[9]

**Data Integrity and Security:** using a server-side component like the GameController on a website enables verification and sanitization of the inputs, which can help maintain data integrity and avoid SQL injection attacks or other additional forms of data manipulation. If the game is directly connected to the database, a user could potentially gain access to the database credentials or manipulate data directly, either by circumventing the game client or by intercepting its traffic. (Figure 18)

**Maintenance and updates:** Each game client would need to be updated whenever the database is modified in any way. This approach avoids redundant code by handling the database interactions only in one place, on the ASP.NET website component, favoring faster game development and updates.

For Stellar Strike, game controller has been built at the site's end into ASP.NET Core MVC Framework to handle asynchronous Unity WebRequests. This controller retrieves the requests and processed the enclosed data. In particular, it obtains the player score from Unity WebRequests and loaded it to SQL Server's "BoardItem" table.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 18 - GameController

The application is able to monitor user scores and retain ongoing ranking through this process of transferring data from the game to the web server. This is a leading example of Unity's ability to interact with server technologies, giving users dynamic and interactive experiences.

The "GameController" class provides an endpoint for submitting game scores to the server. Based on the outcome of the submission process, it will perform needed validations and database functions and respond with appropriate HTTP status codes and JSON payloads. The class uses the [Authorize] tag which allows it to be accessed only by the logged users. If the user is not logged in, they will be redirected to the log in page.

A picture containing text, screenshot, software

Description automatically generated

Figure 19 – GameController Async Post Method

The Post method in this controller is the most important component of the class as it handles the request created by the game. The received score is checked for duplicate values in the database, and it is assigned to the logged user. This method allows the database to store each player’s highscore in order to display it in the leaderboard. However, this method will deal with errors during the process, such as database errors or exceptions. It logs appropriate error messages to the console, indicating that the score submission failed. It then returns a StatusCode response (HTTP 500 Internal Server Error) with a JSON payload containing a "success" property set to false and an error message. (Figure 19)

The game can securely transmit the player's score to the server, verify duplicates, associate scores with logged In users and store them in a database by implementing this "Post" method. This allows the game to display each player's high score in the scoreboard, offering a competitive element and enhancing the overall gaming experience.

## Game Core Algorithms

One of the main, if not the most important class of the game, is the CollisionState.cs class, which is assigned to every object in the game that the user interacts with: player ship, enemies, bullets, cargo pickups etc. It handles how the player’s ship interacts with the other prefabs in Unity by initializing at first the game settings, which are composed of: (Figure 20)

* **CargoSettings:** Handle how and which pickups are dropped when an enemy is destroyed;
* **AdditionalSettings:** Handle the SpriteRenderer and assigns a material type to the asset;
* **EventSettings:** Handle the events loaded when the object is hit. (e.g. enables camera shake when the player is hit/destroyed);
* **OutroSettings:** Handle the methods used for destroying an object. (play explosion sound, show explosion animation, etc.)

A screen shot of a computer program

Description automatically generated with medium confidence

Figure 20 - CollisionState.cs

A series of actions related to destroying objects, counting bodies, triggering outro, and releasing cargo settings will be carried out by the "Kill" method in the code snippet below. It also includes specific behavior for the destruction of the player character, involving score submission to the server and transitioning to the game over scene.

A series of actions related to destroying objects, counting bodies, triggering outro, and releasing cargo settings will be carried out by the "Kill" method in the code snippet below. It also includes specific behavior for the destruction of the player character, involving score submission to the server and transitioning to the game over scene. (Figure 21)

A picture containing text, screenshot, software

Description automatically generated

Figure 21 - Kill() Method

The main aspects of the Kill method, to be more specific handle the following tasks:

* **Invoking the OnOutro() method:** It triggers the explosion animation, the explosion audio, and the other OutroSettings;
* **Releasing cargo settings:** When the object is destroyed, the cargo settings are are used to drop a pickup item;
* **Handling destruction behavior:** The code checks if the current object is the player character, marked as "Player (Defenstar Style)":
  + If it is the player character, the code retrieves the "ScoreSender" component attached to the player, logs the destroyed object's name and the final score, sends the score to the server using the "SendScore" method from the "ScoreSender" component, destroys the current object, and loads the game over scene using the level manager;
  + If the destroyed object is not the player character, it is simply destroyed. (This applies to enemies, bullets, pickups).

A picture containing text, screenshot

Description automatically generated

Figure 22 - Handling Damage Dealt

(Figure 22) This code block handles collisions between game objects and how they affect each other's structural integrity, and contribute to player scores. The examples of collisions handled by this game can be thought of as: player – enemy bullet ; player – enemy ; player – pickup. Items such as enemy bullets and pickup do not actually have a valid structural integrity, meaning that it is not possible to deal damage to a bullet, or to deal damage to a pickup, although they are identified by the script as having the structural integrity = 1.

If either of the objects has any remaining structural integrity, a collision audio effect is played. The effect may vary depending on the material properties of collided objects, which can also provide an effective sound response to collision.

The following two sections update the player's scoreboard when they destroy an object. It checks whether the object hit has a player index and whether its structural integrity has decreased to 0, indicating that it has been destroyed. If this is so, the player's score will be enhanced by the point value of the destroyed object. This update will be performed on both the current object and the object that collided with it.

The Spawner Class is mainly used for spawning the level 1 types of enemies, which are infinitely spawned at a specific time interval (currently set to 8 seconds), as the only way to end the game is by getting the player ship destroyed. (Figure 23)

The Spawner class define several publicly variable types modifiable within the Unity editor. It includes starting position, counter, timer, interval, and intervalRange, which determines the spawn position, number of spawns, interval between spawns, etc.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 23 - Enemy Spawner Class

Among the most important tasks the class can handle are the found in the following methods used:

* **OnSpawn() :** is a public method that can be overridden. This function generates an instance of a random prefab from the prefabs[] array at the specified position and rotation, this means that this type of spawner can be used to spawn different types of enemies (the game’s current design implements this type of spawner only for the level 1 enemies). It then configures several components on the new instance, such as Register, ProgressCounter, EnemyAI, OffCamera, BulletSpawner, and others based on the settings of the spawner.
* **OnSequencerUpdate():** function is overridden to update the spawn timer, spawning new objects when the timer reaches zero, and then resetting the timer. The line which decrements the counter is commented, so that the script spawns an unlimited number of enemies.
* **\_GetPosition():** function sets the position of the spawned object randomly outside the screen, making it unpredictable for the user to anticipate where the next enemy is coming from.

There is also a second type of spawner “ScoreBasedSpawner”, it spawns enemies based on the actual score the player has. This code is used for the more advanced enemies (mini-bosses, bosses, etc.). The score-based spawner’s properties are handled in the following code block: (Figure 24)

A picture containing text, screenshot, software

Description automatically generated

Figure 24 - ScoreBasedSpawner Implementation

The EnemyAI class is used to control the behavior of enemy entities in the game. The behavior can be modified through the Unity Editor or by changing the mode enum directly in the code. This class allows for many different types of enemy behavior to be changed to provide different challenges or to be adapted to different game scenarios. (Figure 25)

The enemy will find random targets from the players group if it does not have a target or its current target is not active in every frame within the Update method. The behavior will then be updated in accordance with the chosen mode.

Each AI mode has a different behavior that could be used for the enemies. The types of AI used in the class are:

* **Cruiser:** This mode will allow the enemy to travel autonomously within a specific range of speed referred to as MinSpeed and MaxSpeed. The enemy is changing direction every second, according to minReflex and maxReflex. The enemy will try to move towards its target if ignoreTarget is set to false. If rotateTowards is true, the enemy rotates toward its moving direction. The friction causes the speed to decrease over time. For infinite spawning enemies, this type of AI is used.
* **HomingMissile:** The enemy will act like a homing missile in this mode, constantly aligning its forward direction to the target and moving in that direction. There is an intro time during which the speed decreases from introSpeed to speed (defined by friction). After the intro time, the enemy will rotate toward the target and move in that direction.
* **Tailer:** The enemy will follow its target directly, moving towards it at a constant speed.
* **Magnet:** This mode is designed to imitate a magnet's behavior. The enemy, according to its polarity, will be attracted or repelled from the target. The range and friction also affect the speed of the enemy.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 25 - EnemyAI Class

The AI behavior may be modified through the Inspector class, which displays each mode's properties for editing in Unity Editor. Modifying the parameters in the editor can apply the changes to the serialized object, as well. This class can be used in various game scenarios and provides a flexible system for controlling enemy behavior. It is an excellent example of classes and inheritances in "Stellar Strike!" that modularise basic AI behavior.

# Web Application & Game Design Approach

## Web Application Flows

A picture containing text, screenshot, diagram, line

Description automatically generated

Figure 26 - Web Application General Process Flowchart

The general process flowchart outlines the user navigation process within the Unity 2D Game integrated with the ASP.NET Core website: (Figure 26)

1. **Start:** The user journey begins with the start, representing the initial interaction of the user with the website.
2. **Home:** The user lands on the Home page, which serves as a central hub leading to different functionalities.
3. **Register:** If the user is new, they can click on the Register option to create a new account.
4. **Home (Post Registration):** Once the registration process is completed, the user is redirected back to the Home page.
5. **Login:** To access the game, the user clicks on the Login option, authenticates themselves by entering their registered credentials.
6. **Play!:** Upon successful login, the user is granted access to the Play! page where the Unity 2D game is located.
7. **Leaderboard:** After finishing the game, the user is directed to the Leaderboard page, which displays the rankings of players based on their game scores.
8. **Home (Post Gameplay):** From the leaderboard, the user can navigate back to the Home page to either play the game again or explore other functionalities of the website.

In addition to this, the user can directly access the 'Leaderboard' from the Home page without having to play the game, providing them a glimpse of player rankings at any point of their navigation.

A picture containing screenshot, diagram, text, line

Description automatically generated

Figure 27 - Detailed Workflow Chart

For the detailed workflow chart, it begins as well with the Start point, indicating when a user opens the application. A decision node “User Registered?” is encountered, where the system checks whether the user is already registered. If the user is not registered, they are directed to the Register operation, where they sign up for the service. This then leads to the Registration Controller, which is responsible for handling users' registration process and keeping their details in a database. (Figure 27)

Users may continue to Login once their registration has been completed or if they have already registered. The Login Controller shall manage the login process, verifying user credentials against the database. After successfully logging in, the user will be directed to the “Play!” view page, where the game can be found.

During gameplay, the Game Controller handles user interactions and game progression. The game data, including user scores, is continuously updated and stored in the database.

The system will trigger a "Game Over" node as soon as the gaming session is over. Depending on the user's score and rank, the Leaderboard is updated with the latest statistics, showcasing top performers within the gaming application. This will be carried out by using the "Leaderboard Controller" service, which retrieves the required information from the database.

This workflow chart effectively captures various components of the application, such as user registration, login, game controller, and leaderboard updates. This also shows how users and the game data are being managed and stored within those components as well as the crucial role of a database. The chart provides an overview of the user journey, the backend processes, and the decision points within the application, which can be beneficial for understanding the system's operation and for future enhancements.

## Web Application Views/Pages

A screenshot of a video game

Description automatically generated

Figure 28 - Home Page

The home page provides an engaging and dynamic user experience, specifically developed for the "Stellar Strike" game. A carousel, which is an automated slide show that occupies the whole width of the page and contains a series of five individually selected slides providing an interactive overview of each aspect of the game, has been chosen as the most important visual element in this web page. (Figure 28)

Home page’s layout is set against a visually stunning background of a star-studded cosmos. The backdrop of this space theme, which injects a sense of mystery and exploration into the entire website, maintains consistency in all its pages. An elegant, neutral canvas that occasionally glittered with gold and neon greens for emphasis and interactivity is made up of grey tints in the animated star background.

The carousel items could be briefly described in this order as follows:

1. First slide is a welcoming introduction to "Stellar Strike!";
2. Second slide displays a 3-step, concise "Game Overview";
3. Third slide shifts focus to the "How to Play" portion of the game;
4. Fourth slide showcases vivid "Game Screenshots", arranged neatly in a three-column grid;
5. Fifth slide features the "Contact Information" ff the developer.

The Home page has an overall role of providing a mean to access all the features and other pages of the website, through the navbar implemented.

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure 29 - Register Page

The "Register" page on the "Stellar Strike" game site delivers a streamlined and user-friendly registration process while maintaining the consistent theme of a starry cosmos in the backdrop. The page layout is perfectly balanced between functionality and guidance for the user, neatly divided into two columns. (Figure 29)

About one-third of the page width is devoted to the left column with a registration form in it. This form begins with a succinct headline, "Create a new account," followed by a separator line that helps distinguish the headline from the form fields. The following are four 'form floating' sections, each with its own input fields: email, username, password, and confirm password. The placement of these sections gives a structured, orderly appearance to the form, facilitating effortless navigation for the user.

An appropriate label, which helps to identify the information required, shall be displayed on each input field in these sections. In addition, a real-time validation feature will be added to each field. Any errors or omissions are highlighted with text in a bold, red font, ensuring immediate feedback to the user and thus an efficient correction of any input mistakes.

As a supplementary guide, the right column occupies approximately half the width of the page, and provides a helpful password tip. A distinctive heading and a separator line follow the beginning of this section. The importance of the password security is then explained in a brief, helpful paragraph that provides useful information to users on how to create strong and unique passwords. This information is encapsulated in a slightly informal, conversational tone, making it easy to comprehend.

In general, the Register page is effective in communicating its purpose and offering users an easy, user-friendly interface for creating a new account and immersing themselves in Star Strike's universe.

A screenshot of a computer

Description automatically generated

Figure 30 - Login Page

The "Log in" page for the "Stellar Strike" game is characterized by an intuitive layout and a thematic continuity that resonates with the rest of the site. The page shows the familiar celestial backdrop of the starry sky, setting the stage for an adventurous journey into the game's universe. (Figure 30)

There is a neat split of two columns in the layout. The login form is contained in a left column, which occupies about three-thirds of the width. It starts with an easy headline: "Log in to your account," followed by a separate line that draws the user's attention to the task. The form consists of two 'form-floating' sections for Username (or Email) and Password. The real-time validation feature, indicated by text in a bold, red font, enhances user experience by providing instant feedback on any erroneous inputs.

Furthermore, this form has three hyperlinks for quick access to secondary actions. Users can retrieve forgotten passwords, register as a new account, or resend email confirmations, which are helpful options to avoid any problems during the login process.

The right column occupies roughly half of the width, and carries an enticing message under the headline "Embark on an Extraordinary Journey". This engaging call to action, teasing the thrilling experience that awaits the player, serves to entice users to log in and immerse themselves in the game.

Overall, the "Log In" page achieves its purpose by providing a simple and visually appealing interface that enables users to easily transition to the "Stellar Strike" universe.

A screenshot of a video game

Description automatically generated

Figure 31 - Play! Page

The "Play!" page of "Stellar Strike" sets the stage for the gaming experience, featuring a seamless blend of an enticing space-themed design and high-performing game technology. The page displays a star-studded backdrop featuring shades of grey, gold, and hints of neon green. (Figure 31)

There is a dominant game screen in the middle of this page, which resides within a "unity-container" division. The Unity WebGL player allows the game to be played on any canvas, allowing it to act as a launch pad for exciting intergalactic journeys. To maintain a visually satisfying balance between the game interface and the fascinating background, for users of desktop computers, the canvas is scaled to represent 70% of the viewport's height and width.

The footer of the canvas houses three crucial elements - the WebGL logo, a fullscreen button, and a text message encouraging players to try fullscreen mode. The fullscreen button is interactive and allows users to experience the game in a truly immersive fullscreen mode when they click it.

The script dynamically loads the necessary Unity engine files and creates the game instance. During the loading process, the script adjusts the full progress bar width based on the progress, giving users a clear indication of the loading status. Once the loading is complete, it hides the loading bar and enables the Fullscreen button, allowing users to maximize their game window.

The "Play!" page masterfully balances aesthetic design and game functionality, creating an environment that invites players to immerse themselves in the captivating world of "Stellar Strike."

A screenshot of a video game

Description automatically generated

Figure 32 - Leaderboard Page

An exciting display of competitive spirit that reflects the game's best players is displayed on the Leaderboard page for "Stellar Strike." The page continues the theme of space, and visitors are immersed in a galactic atmosphere characterized by a star-studded backdrop in shades of grey with a hint of gold and neon green. (Figure 32)

The Leaderboard, a critical component that provides an atmosphere of challenge and development among players, is at the heart of this page. The Leaderboard has been divided into its own "leaderboard container" division, which is a scrollable table. This will make it easy for players to navigate the whole Leaderboard even if they have ten or a hundred entries. The table is presented with a sticky header, ensuring that the three column names, "#" (representing the player's rank), "Username," and "Score," remain visible as the user scrolls through the list.

Each row corresponds to an entry on a leaderboard for one player in the table's body. This list includes a player's rank, which will be dynamically set based on their position in that list, username, and score. All this data is fetched from the LeaderboardController, demonstrating the seamless integration of the game's backend data with its frontend display.

In summary, the Leaderboard area is a perfect place for players to check their status in competition with other players, enhancing the competitive spirit of 'Stellar Strike.' The page creates a unique theme for space that complements the clear, easy-to-read scoreboard and provides another layer of excitement in gameplay.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 33 - Privacy Policy Page

The Privacy Policy page for "Stellar Strike" is designed with a strong emphasis on clarity and user understanding. In order to make it easy for users to understand the company's practices concerning their data, its clean and organized layout, together with a simple language, helps. The page increases readability and facilitates easy navigation through different policy sections, based on minimal design principles with a dedicated privacy.css style sheet. (Figure 33)

## Game Design Brief

### Game Overview

The game is titled "Stellar Strike," and it is a space-themed shooter game. The game will run in three states, namely: Main Menu, Gameplay, and Game Over.

### Game States

1. **Main Menu**

This is the initial state of the game. The title of the game, "Stellar Strike," is prominently displayed, along with a "Start" button to begin gameplay. (Figure 34)



Figure 34 - Main Menu Screen

1. **Gameplay**

The player will be transferred to the Gameplay state when they click the "Start" button. The player can navigate his ship with their mouse and shoot it by clicking the left mouse button in this state. The gameplay includes three tiers of enemies, which will spawn depending on the player's score. Other pickups that can help the player in various ways will also be dropped by the enemy. (Figure 37)

The game area is a 2D space that is constantly filled with enemy spacecrafts of different tiers. The difficulty level and the type of threat that they pose to the players is represented by their enemy tier: (Figure 35)

* **Tier 1 Enemies:** These are the basic enemies that will spawn infinitely throughout the game.
* **Tier 2 Enemies:** These enemies are more advanced and will start spawning once the player reaches a score of 1000.
* **Tier 3 Enemies:** The most formidable enemies, these will start spawning once the player reaches a score of 2500.



Figure 35 - Player & Enemies

In order to ensure the game is challenging and entertaining, enemies will be instructed to pop up and move in different formations and patterns. They can drop various pickups when destroyed, which can significantly help the player's survival: (Figure 36)

* **Health:** Restores a certain portion of the player's health.
* **Shield:** Activates a protective barrier around the player's spaceship, making it invulnerable until the shield is destroyed..
* **Laser:** This is a special weapon pickup. It targets and destroys incoming enemy bullets, creating a safe space for the player.
* **Drone:** An extra firepower pickup. When collected, it summons an automatic shooting drone that hovers around the player's ship and fires towards enemies.

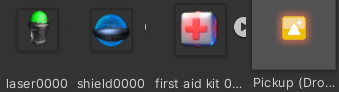


Figure 36 – Pickups

When playing, the score will be shown at the top of the screen with updates constantly as players destroy enemies. The players' health is displayed in the top leftmost corner, decreasing when their spaceships are hit by enemy fire.

A screenshot of a video game

Description automatically generated

Figure 37 - Gameplay Screen

The game will continue until the player's health bar is fully depleted, leading to the Game Over state. The transition between game states should be smooth and intuitive, ensuring the player knows the shift and the new objectives or outcomes associated with the new state.

1. **Game Over**

If a player's health is down to zero, this state will be activated. A game over screen will display the player's final score, and the buttons to restart the game or return to the main menu. (Figure 38)

A screenshot of a video game

Description automatically generated

Figure 38 - Game Over Screen

### Target Audience

“Stellar Strike!” aimed at players who enjoy space shooters and fast action games. With its various tiers of enemies and the diversity of pickup, it aims to give players a challenge and thrilling experience.

### Artistic Direction

The game is going to be retro and spacey in style. Rather than visual complexity, the graphics should be clean and straightforward, focusing on the game's mechanics. It uses a dark background to display the vastness of space and bright colors for players and foe sprites so that they stand out against the backdrop. In order to enhance the overall gaming experience, the sound design will include engaging sound effects for shooting, enemies spawning, pickups, and a compelling soundtrack.

### Platform

The game will be developed as a web game, ensuring accessibility across different platforms (PC, Mac, Linux) and devices (desktop, laptop, tablet) with modern web browsers.

# Bibliography

[1] ardalis, “Overview of ASP.NET Core MVC,” Jun. 27, 2022. https://learn.microsoft.com/en-us/aspnet/core/mvc/overview (accessed Jun. 29, 2023).

[2] “Gaming Market Size, Share & Trends Analysis Report, 2030.” https://www.grandviewresearch.com/industry-analysis/gaming-industry (accessed Jun. 29, 2023).

[3] T. Brookes, “Adobe Flash is Dead: Here’s What That Means,” *How-To Geek*, Jan. 01, 2021. https://www.howtogeek.com/700229/adobe-flash-is-dead%c2%a0heres-what-that-means/ (accessed Jun. 30, 2023).

[4] “Complete guide to ASP.NET Core MVC (.NET 8) [E-Commerce App],” *Udemy*. https://ea.udemy.com/course/complete-aspnet-core-21-course/ (accessed Jun. 30, 2023).

[5] A. Freeman, *Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages*. Berkeley, CA: Apress, 2022. doi: 10.1007/978-1-4842-7957-1.

[6] B. Kaliski, “PKCS #5: Password-Based Cryptography Specification Version 2.0,” Internet Engineering Task Force, Request for Comments RFC 2898, Sep. 2000. doi: 10.17487/RFC2898.

[7] U. Technologies, “Unity - Manual: Deploy WebGL application.” https://docs.unity3d.com/Manual/webgl-deploying.html (accessed Jun. 29, 2023).

[8] “C# Unity Developer 2D Coding: Learn to Code Video Games,” *Udemy*. https://ea.udemy.com/course/unitycourse/ (accessed Jun. 30, 2023).

[9] U. Technologies, “Unity - Scripting API: UnityWebRequest.” https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html (accessed Jun. 29, 2023).