# Asteroid Shooter GUI Project Documentation

GitHub: <https://github.com/JayMackay/AsteroidShooterGUI>

Trello Board: <https://trello.com/b/xP2y4Xa7/asteroid-shooter-application>

# SPRINT ZERO

*Project Planning*

Initial plan to create a WPF application with user sign up and login to save user high scores and have a private account to store data via SQL utilizing Entity Framework.

*Project Parameters*

* User login page
* Registration page
* Three mini games (Snake, battleship & asteroid shooter)
* SQL backend database with user & high score table

*Users Stories*

* Sign up & login with account and credentials
* Each game stores the high score based on the linked user
* Transitioning between windows should be simple and easy with buttons clearly labelled
* Clean & modern UI design
* Game state can be paused

***PRODUCT BACKLOG***

* Main window login UI
* Registration form UI
* SQL database & Entity Framework integration
* User registration submit functionality
* Login request database functionality
* Game window UI
* Game logic
* High score request & submit functionality

# SPRINT ONE

*Sprint Backlog*

* Login window UI
* Database model
* Entity framework initialization

*Review*

Managed to create a clean and elegant login UI which matched the requirements that I set. The task was finished quite quickly so ended up creating a database model utilizing the DbContext class with Entity Framework to set up a migration and initialize the tables. As a blocker I did have an interview scheduled for the following day, which I did allocate most of my time preparing. Although, this was quite minor.

*Retrospective*

* Stop getting sidetracked with unnecessary functionality this could be refactored later down the line
* Keep timeboxing each task and breaking it down into easier to manage components
* Take breaks don’t continuously work for long periods as it is unproductive, and you will tire out easily

# SPRINT TWO

*Sprint Backlog*

* Registration window UI
* Registration SQL functionality

*Review*

UI design was simple and straight forward, I copied over the login window design while adding additional elements in relation to a registration form. Implemented basic validation methods for user input verification. Hard coded SQL data manipulation queries to insert values into the SQL database once the user hits submit. Due to my inexperience with Entity Framework I did spend a lot of time researching how to combine my C# application with SQL and how to go about implementing the intended functionality. Furthermore, I felt it was quite ambitious to create three separate games as I overestimated the time it would take to complete the project and so I reduced it down to one game only.

*Retrospective*

* When experiencing a blocker try to set an allocated amount of time to complete the task
* If the task is not completed within that time limit move on to the next, to be more efficient

# SPRINT THREE

*Sprint Backlog*

* Login Window Functionality
* Game Window

*Review*

Utilizing a similar structure for the SQL registration functionality, I created a method that extracts user credentials from the database using standard SQL data queries, again this was hardcoded into the XAML C# file. This would link to a new window that matched the credentials with the name of the user to display a string. After asking Cathy for advice she recommend that I refactor my code based on the following:

* Create a middle layer class to save new user to send the parameters of the method as strings to query the database LINQ query
* Set separate methods for each query of the application
* This in turn separates different areas of functionality

*Retrospective*

* Start writing unit tests based on the application functionality

# SPRINT FOUR

*Sprint Backlog*

* Game Window UI
* Game Logic

*Review*

Imported game assets for the sprites of the game objects. Implemented some basic functionality utilizing system threading as a game loop. Initialized the objects within the game using scripts to create instances of each object such as the player, enemies and bullets. Implemented variables for movement as well as initializing a canvas for the game background.

# SPRINT FIVE

*Sprint Backlog*

* Game Engine Logic
* Garbage Collection
* Minor UI Fixes

*Review*

Realized there were some minor UI bugs when displaying the score and the users name. Added key input functionality on key press and release. Implemented a list to remove unused generated objects. Implemented a separate class with similar methods to the XAML class to create unit tests based on functionality.

# PROJECT RETROSPECTIVE

During initial planning the scope and size of my project was quite large with multiple features. Due to inexperience I did not take into consideration the time frame and the ease of execution in order to get all my planned functionality completed. In the future I aim to start with a simple project and look to add features nearer the end of the project once I have a minimum viable product.

Furthermore, I focused most of my time on GUI design over general functionality. As a result, all my methods were hard coded into my XAML C# classes. In terms of structure I know this is not as convenient and maintainable.

In the future I will follow a more structured approach. Rather than start with the GUI I will start creating multiple classes such as ***Game Logic*** as well as a ***Middle Laye***r with all my SQL queries. I can get the hardest aspect of the project out of the way before focusing on added functionality and the UI.

To further iterate, I found that integrating Entity Framework into my project was the most difficult part of the whole development process. However, this can be down again to inexperience, but by timeboxing and breaking it down into easier to manage steps I could have possibly overcome the blocker quicker rather than spending an entire sprint trying to resolve my issues.

Finally, I found by coding all the functionality and game logic in the main methods it was difficult to create unit tests for the project and it further hindered my progress. Instead I had to create a separate ***Test Method*** class with all the functions needed for testing, creating unnecessary work.