# SE 3XA3: Test Plan
# Space Invaders

March 11, 2022

Team Information:

| Team Number | Name | MACID |
|---|---|---|
| L03 G07 | Qianlin Chen | chenq84 |
| | Jiacheng Wu | wuj187 |
| | Tingyu Shi | shit19 |

Table 1: Revision History

| **Date** | **Developer(s)** | **Change** |
|---|---|---|
| January 26, 2022 | All team members | Initial Document |
| March 10, 2022 | Qianlin Chen | Test for functional req and nonfunction req |
| March 10, 2022 | Jiacheng Wu | General Information, Plan and matrix |
| March 10, 2022 | Tingyu Shi | Tests for Nonfunctional req, section 4 - 7 |

# Contents

# List of Tables



# List of Figures

# 1 General Information

## 1.1 Purpose

The test plan describes the detailed procedures of our testing towards the functional and non-functional requirements of our project "Space Invaders". The purpose of this document is to apply the methods and plans after the project is implemented in order to minimize the chance the users will see errors and build the confidence the project will be working properly.

## 1.2 Scope

The document will include the plans for testing, all the tests for functional and non-functional requirements in the SRS document and tests of proof of concept. After that, the document will include the plans for unit tests. The document will be revised as the project is developed and modified.

## 1.3 Acronyms, Abbreviations, and Symbols

Table 2: Table of Abbreviations

| **Abbreviation** | **Definition** |
|---|---|
| SRS | Software requirement specification |
| PC | personal computer |
| FPS | frams per second |
| GUI | graphic user interface |

Table 3: Table of Definitions

| **Term** | **Definition** |
|---|---|
| Space invader | The name of a video game |
| Graphic User Interface | a graphic representation for users to interact |
| Score | The number that represents the achivement of users |
| Players | the users of the game that follow certain rules to succeed |
| Aircraft | the users will controll the aircraft to play the game |
| Monster | the goal of the game, the target the aircraft to attack. |
| Ammo item | the element in game that can boost users' ammo |
| Health item | If the aircraft shoots a Health item, the aircraft will gain one more health point. |
| Bomb item | If the aircraft shoots a Bomb item, the bomb will explode and do damage to a big range of the area in the monster matrix. |

### 1.4 Overview of Document

The plans for testing will include the description of the software, the team to test it, the approach to automated testing, the tools for testing and the schedule for testing. After that, it will go through all the detailed tests for requirements specified in SRS along with the proof of concept.Lastly we will compare with the original implementation and discuss the unit testing plan.

## 2 Plan

### 2.1 Software Description

Space Invaders is a fixed shooter in which the player moves aircraft horizontally across the bottom of the screen and fires at monsters overhead. The original repository was using pygame and python to implement. We will use python and pygame to implement it and apply the software principles to redevelop the game. Additionally, we will upgrade the graphic user interface of it.

### 2.2 Test Team

The core test team consists of all members of Group-07 are responsible for writing and executing tests:

- Tingyu Shi
- Qianlin Chen
- Jiacheng Wu

### 2.3 Automated Testing Approach

In this project, we mainly use the exploration test since we will test the display and controller and the tests of those depend on the input of the users.However, the unit test will be one of our automated testing approach to test our models. Models include the following:

- Monster
- MonsterMatrix
- SpaceShip
- Score
- Bullets

Since the game is still in the process of implementation, more models may be created in the future.

### 2.4 Testing Tools

The testing tool we are going to use is Pytest. It's a testing tool used for unit test for python programs.

## 2.5 Testing Schedule

See Gantt Chart at the following url:

- .gan File
- .pdf File

# 3 System Test Description

## 3.1 Tests for Functional Requirements

The software architecture used for this game is MVC. As a result, we have three testing areas, which are model, view and control.
NOTE: The game implementation is not finished yet, so you may not be able to find some of methods mentioned below in our source code.

### 3.1.1 Model

1. Test-FR15-M1
   Type: Unit test (functional, dynamic, automated)
   Initial State: Bullet from the aircraft hits one of the monsters.
   Input/Condition: `getScore()` is called on `Score` object.
   Output/Result: Player's score increase.
   How test will be performed: Testers can check whether the score goes up once they hit monsters by comparing with the original score.

2. Test-FR16-M2
   Type: Unit test (functional, dynamic, automated)
   Initial State: Monsters' bullets hit aircraft.
   Input/Condition: `getLives()` is called on `SpaceShip` object.
   Output/Result: Aircraft's lives decrease.
   How test will be performed: Testers can check whether the number of lives goes down once the aircraft is hit by bullets from monsters.

3. Test-FR16-M3
   Type: Unit test (functional, dynamic, automated)
   Initial State: Bullets from aircraft hit the health game item in the monster matrix.
   Input/Condition: `getLives()` is called on `SpaceShip` object.
   Output/Result: Aircraft's lives increase.
   How test will be performed: Testers can check whether the number of lives goes up once bullets from aircraft hit the health game item in the monster matrix.

4. Test-FR25-M4
   Type: Unit test (functional, dynamic, automated)
   Initial State: A green monster is hit by one bullet from the aircraft.
   Input/Condition: `isDead()` is called on `Monster` object.
   Output/Result: `isDead()` should return true.
   How test will be performed: Testers can check if the `isDead()` returns true or just observe whether the green monster disappears in the monster matrix or not if the green monster is hit by one bullet from the aircraft.

5. Test-FR26-M5
   Type: Unit test (functional, dynamic, automated)
   Initial State: A blue monster is hit by two bullets from the aircraft.
   Input/Condition: `isDead()` is called on `Monster` object.
   Output/Result: `isDead()` should return true.
   How test will be performed: Testers can check if the `isDead()` returns true or just observe whether the blue monster disappears in the monster matrix or not if the blue monster is hit by two bullets from the aircraft.

6. Test-FR27-M6
   Type: Unit test (functional, dynamic, automated)
   Initial State: A pink monster is hit by three bullets from the aircraft.
   Input/Condition: `isDead()` is called on `Monster` object.
   Output/Result: `isDead()` should return true.
   How test will be performed:Testers can check if the `isDead()` returns true or just observe whether the pink monster disappears in the monster matrix or not if the pink monster is hit by three bullets from the aircraft.

7. Test-FR29-M7
   Type: Unit test (functional, dynamic, automated)
   Initial State: One bullet hits an obstacle.
   Input/Condition: `getArea()` is called from the `Obstacle` object.
   Output/Result: The return value from the above method call should be less than the previous obstacle area.
   How test will be performed: Testers can call `getArea()` on `Obstacle` object and record the return value as $S_1$. Then the tester can make the aircraft to shoot one bullet toward the obstacle and then call `getArea()` again and record the return value as $S_2$. Finally, make sure that $S_2 < S_1$.

8. Test-FR30-M8
   Type: Unit test (functional, dynamic, manual)
   Initial State: Game GUI presents one game item, one monster and one aircraft.
   Input/Condition: Monster shoots one bullet to the game item firstly and aircraft shoots one bullet to the game item secondly.
   Output: The game item should only disappear after being hit by the bullet from the aircraft.
   How test will be performed: Testers can set the game GUI with just three elements, which are a monster, a game item and a aircraft. Let the monster and the aircraft shoot bullets to the game item alternatively. The game item should only disappear after being hit by the bullet from the aircraft.

9. Test-FR31-M9
   Type: Unit test (functional, dynamic, manual)
   Initial State: Start a new game round.
   Input/Condition: No inputs will be given for this test case. However, testers should record the number of game items presented in the monster matrix.
   Output/Result: The number of game items in each game round should be less than 5.
   How test will be performed: Testers can start a new game round multiple times and record the number of game items in each game round. After that, calculate the average number of game items presented in one game round. The result should be less than 5.

10. Test-FR32-M10
    Type: Unit test (functional, dynamic, manual)
    Initial State: GUI displays a monster matrix with a bomb game item in it.
    Input/Condition: Testers operate the aircraft and shoot a bullet from the aircraft to the bomb game item.
    Output/Result: 8 monsters around the bomb game item should disappear after the bomb is hit.
    How test will be performed: Testers set a game round with just one bomb game item in the monster matrix and then operate the aircraft to shoot the bomb game item.

11. Test-FR33-M11
    Type: Unit test (functional, dynamic, manual)
    Initial State: GUI displays a monster matrix with a ammo game item in it.
    Input/Condition: Testers operate the aircraft and shoot a bullet from the aircraft to the ammo game item.
    Output/Result: The number of bullets can be shot from the aircraft should increase 1.
    How test will be performed: Testers set a game round with just one ammo game item in the monster matrix and then operate the aircraft to shoot the ammo game item.

12. Test-FR34-M12
    Type: Unit test (functional, dynamic, manual)
    Initial State: GUI displays a monster matrix with a heart game item in it.
    Input/Condition: Testers operate the aircraft and shoot a bullet from the aircraft to the heart game item.
    Output/Result: Aircraft's lives should increase 1.
    How test will be performed: Testers set a game round with just one heart game item in the monster matrix and then operate the aircraft to shoot the heart game item.

13. Test-FR35.1-M13
    Type: Functional, Dynamic, Manual
    Initial State: Several monster matrices are loaded.
    Input/Condition: No inputs will be given in this test case. Testers need to make a screenshot for each monster matrix.
    Output/Result: Game items should occur randomly in different matrices.
    How test will be performed: Testers can load several monster matrices and make a screenshot for each matrix. After that, testers can analyze screenshots to ensure the random occurrence of game items.

14. Test-FR39-M14
    Type: Unit test (functional, dynamic, manual)
    Initial State: Start a new game.
    Input/Condition: Testers shoot bullets from aircraft by press `SPACE` key.
    Output/Result: The number of bullets should be 1.
    How test will be performed: Testers start a new game and shoot bullets by press `SPACE` key.

### 3.1.2 View

1. Test-FR1-V1
   Type: Functional, Dynamic, Manual
   Initial State: Welcome message has been displayed.
   Input/Conditin: Testers enter any key.
   Output/Result: The GUI should display a message and let testers to choose the play mode.
   How test will be performed: Testers should press any key after the welcoming message has been displayed.

2. Test-FR2-V2
   Type: Functional, Dynamic, Manual
   Initial State: Game mode selection UI has been displayed.
   Input/Condition: Testers enter `S` or `D` to choose the play mode.
   Output/Result: GUI should display three monster types and corresponding scores.
   How test will be performed: Testers should press `S` or `D` to choose the play mode and then check if three monster types and corresponding scores can be displayed.

3. Test-FR3-V3
   Type: Functional, Dynamic, Manual
   Initial State: Game mode selection UI has been displayed.
   Input/Condition: Testers enter `S` or `D` to choose the play mode.
   Output/Result: GUI should display game instructions.
   How test will be performed: Testers should press `S` or `D` to choose the play mode and then check if game instructions can be displayed.

4. Test-FR4-V4
   Type: Functional, Manual, Dynamic
   Initial State: Game instructions and introduction to monsters have been displayed.
   Input/Condition : Testers enter the game by pressing any key.
   Output/Result: The game should display the total score at the left up corner and the initial score should be 0.
   How test will be performed: This is really an exploratory test. Testers can play the game and the score information should always be available at the left up corner.

5. Test-FR5-V5
   Type: Functional, Manual, Dynamic
   Initial State: Game instructions and introduction to monsters have been displayed.
   Input/Condition: Testers enter the game by pressing any key.
   Output/Result: The game should display total 5 lives at the beginning of the game for the aircraft.
   How test will be performed: Testers can enter the game by pressing any key after reading the game instruction and introduction about the monsters.

6. Test-FR6-V6
   Type: Functional, Manual, Dynamic
   Initial State: Game instructions and introduction to monsters have been displayed.
   Input/Condition: Testers press any key to enter the game.
   Output/Result: The game should display a galaxy picture as the background.
   How test will be performed: A tester can check if a galaxy picture is displayed in each game round as the background.

7. Test-FR7-V7
   Type: Functional, Manual, Dynamic
   Initial State: Game instructions and introduction to monsters have been displayed.
   Input/Condition: Testers enter to the game by press any key.
   Output/Result: The game should display a monster matrix(5 rows and 10 columns).
   How test will be performed: Testers can check if a monster matrix(5 rows and 10 columns) is displayed in each game round by playing the game.

8. Test-FR8-V8
   Type: Functional, Manual, Dynamic
   Initial State: Game instructions and introduction to monsters have been displayed.
   Input/Condition: Testers press any key to enter the game and then make a screenshot of the monster matrix.
   Output/Result: Screenshot about monster matrix.
   How test will be performed: Multiple testers can start the game and make screenshots about the monster matrix. After that, they can compare different monster matrix screenshots to ensure that game items are displayed randomly in the matrix.

9. Test-FR9-V9
   Type: Functional, Manual, Dynamic
   Initial State: Game mode selection message has been displayed.
   Input/Condition: Testers press `S` to select the single-player mode and then start the game.
   Output/Result: Game GUI should display a single aircraft.
   How test will be performed: Testers can start the game with single-player mode multiple times to check if there is only one aircraft displayed on the screen.

10. Test-FR9-V10
    Type: Functional, Manual, Dynamic
    Initial State: Game mode selection message has been displayed.
    Input/Condition: Testers press `D` to select the double-player mode and then start the game.
    Output/Result: Game GUI should display two aircraft.
    How test will be performed: Testers can start the game with double-player mode multiple times to check if there are aircraft displayed on the screen.

11. Test-FR10-V11
    Type: Functional, Manual, Dynamic
    Initial State: Game instructions and introduction to monsters have been displayed.
    Input/Condition: Testers enter the game by pressing any key.
    Output/Result: The game GUI should display four obstacles between the aircraft and the monster matrix.
    How test will be performed: Multiple testers can check if four obstacles can be displayed in each game round by just playing the game.

12. Test-FR11-V12
    Type: Functional, Manual, Dynamic
    Initial State: Testers finish the game either because of winning the game or failing the game.
    Input/Condition: No inputs will be given for this test case. But testers should check if final score can be displayed.
    Output/Result: The game GUI should display final score. How test will be performed: Testers can check if the final score can be displayed once they finish the game.

13. Test-FR12-V13
    Type: Functional, Manual, Dynamic
    Initial State: Testers are in game round 5.
    Input/Condition: Testers pass game round 5.
    Output/Result: Game GUI should display 'WIN!'
    How test will be performed: Multiple testers should try to pass the game and check if 'WIN!' can be displayed if they pass the game.

14. Test-FR13-V14
    Type: Functional, Manual, Dynamic
    Initial State: Testers are in any game round.
    Input/Condition: Testers lose the current game round.
    Output/Result: Game GUI should display 'FAIL!'
    How test will be performed: Multiple testers can fail any game round on purpose and check if 'FAIL!' can be displayed on the screen.

15. Test-FR14-V15
    Type: Functional, Manual, Dynamic
    Initial State: 'WIN!' or 'FAIL!' have been displayed.
    Input/Condition: No inputs will be given for this test case. Testers should check if welcoming message can be displayed after 'WIN!' or 'FAIL!' have been displayed.
    Output/Result: Game GUI should display the welcoming message.
    How test will be performed: Multiple testers can try to pass the game or fail the game on purpose and then check if game GUI can display the welcoming message after displaying 'WIN!' or 'FAIL!'.

16. Test-FR28-V16
    Type: Functional, Manual, Dynamic
    Initial State: Testers start a game.
    Input/Condition: Testers shoot a green monster once.
    Output/Result: The green monster should disappear.
    How test will be performed: Testers can start a game and shoot green monsters and they should disappear after being shot once.

17. Test-FR28-V17
    Type: Functional, Manual, Dynamic
    Initial State: Testers start a game.
    Input/Condition: Testers shoot a blue monster twice.
    Output/Result: The blue monster should disappear.
    How test will be performed: Testers can start a game and shoot blue monsters and they should disappear after being shot twice.

18. Test-FR28-V18
    Type: Functional, Manual, Dynamic
    Initial State: Testers start a game.
    Input/Condition: Testers shot a pink monster three times.
    Output/Result: The pink monster should disappear.
    How test will be performed: Testers can start a game and shoot pink monsters and they should disappear after being shot three times.

### 3.1.3 Control

1. Test-FR17-C1
    Type: Functional, Manual, Dynamic
    Initial State: Game is not started.
    Input/Condition: Testers enter the game and pass through the whole game and record the number of game rounds.
    Output/Result: The game should contain 5 rounds in total.
    How test will be performed: Testers should continue to play game until it displays 'WIN!' then count the total game rounds they played.

2. Test-FR18-C2
   Type: functional, manual, dynamic
   Initial State: Game instructions and introduction to monsters have been displayed.
   Input/Condition: Testers press any key to enter the first game round.
   Output/Result: Game GUI should display five rows of green monsters.
   How test will be performed: Testers can play game round1 and check if there are five rows of green monsters.

3. Test-FR19-C3
   Type: functional, manual, dynamic
   Initial State: Testers pass the first game round.
   Input/Condition: No inputs will be given for this test case. Once game round 1 is passed, the game will automatically turn to game round 2.
   Output/Result: Game GUI should display 3 rows of green monsters and 2 rows of blue monsters.
   How test will be performed: Testers can play game round2 and check if there are 3 rows of green monsters and 2 rows of blue monsters.

4. Test-FR20-C4
   Type: functional, manual, dynamic
   Initial State: Testers pass the second game round.
   Input/Condition: No inputs will be given for this test case. Once game round 2 is passed, the game will automatically turn to game round 3.
   Output/Result: Game GUI should display 5 rows of blue monsters.
   How test will be performed: Testers can play game round3 and check if there are 5 rows of blue monsters.

5. Test-FR21-C5
   Type: functional, manual, dynamic
   Initial State: Testers pass the third game round.
   Input/Condition: No inputs will be given for this test case. Once game round 3 is passed, the game will automatically turn to game round 4.
   Output/Result: Game GUI should display 1 row of green monsters, 3 rows of blue monsters and 1 row of pink monsters.
   How test will be performed: Testers can play game round4 and check if there are 1 row of green monsters, 3 rows of blue monsters and 1 row of pink monsters.

6. Test-FR22-C6
   Type: functional, manual, dynamic
   Initial State: Testers pass the fourth game round.
   Input/Condition: No inputs will be given for this test case. Once game round 4 is passed, the game will automatically turn to game round 5.
   Output/Result: Game GUI should display 5 rows of pink monsters.
   How test will be performed: Testers can play game round5 and check if there are 5 rows of pink monsters.

7. Test-FR23-C7
   Type: functional, manual, dynamic
   Initial State: Testers are playing in different game rounds.
   Input/Condition: No inputs will be given for this test case. Testers should record the movement track of the monster matrix.
   Output/Result: The monster matrix movement track should be east-south-west.
   How test will be performed: Testers can play different game rounds and record the movement track of monster matrices.

8. Test-FR24-C8
   Type: functional, manual, dynamic
   Initial State: Testers are in different game rounds.
   Input/Condition: No inputs will be given for this test case. Testers should record how monsters shoot bullets.
   Output/Result: Recordings show that monsters shoot bullets randomly in different game rounds.
   How test will be performed: Multiple testers can play the game at the same time and check if monsters shoot bullets in random ways.

9. Test-FR35.2-C9
   Type: functional, manual, dynamic
   Initial State: Game instructions and introduction to monsters have been displayed.
   Input/Condition: Testers press any key.
   Output: GUI shows the game.
   How test will be performed: Testers can see if they can enter to the game successively by pressing any key after game instructions and introduction to monsters have been displayed.

10. Test-FR36-C10
    Type: functional, manual, dynamic
    Initial State: Testers are in any game round.
    Input/Condition: Testers click cross icon at the right top corner.
    Output/Result: Game exited.
    How test will be performed: Testers can see if they can exit the game successively by clicking the cross icon.

11. Test-FR37-C11
    Type: functional, manual, automated
    Initial State: Aircraft is in a stationary state with position (370,350).
    Input/Condition: Testers press $\leftarrow$ or $\rightarrow$
    Output/Result: Aircraft new position.
    How test will be performed: Testers can call `getX()` on the corresponding `SpaceShip` object to see if the x coordinate decreases when pressing $\leftarrow$ or increases when pressing $\rightarrow$.

12. Test-FR38-C12
    Type: functional, manual, dynamic
    Initial State: The second aircraft is in a stationary state with position (370,350).
    Input/Conditino: Testers press `A` or `D`.
    Output/Result: Second aircraft's new position.
    How test will be performed: Testers can call `getX()` on the second `SpaceShip` object to see if the x position decreases when pressing `A` or increases when pressing `D`.

NOTE: There are two `FR35` in our SRS document.(This mistake will be revised in our revision1). The test for the first `FR35` is `Test-FR35.1-M13`. The test for the second `FR35` is `Test-FR35.2-C9`.

## 3.2 Tests for Nonfunctional Requirements

### 3.2.1 Look and Feel Testing

1. Test-NFR1-LF1
   Type: Dynamic, Automated, Functional
   Initial State: Game started.
   Input/Condition: Testers use external tool to measure the FPS.
   Output/Result: FPS value should always be greater than 30.
   How test will be performed: Testers can record the FPS of the game every 10 seconds and measure FPS for 2 minutes. After that, testers can check if all the values recorded are greater than 30.

2. Test-NFR2-LF2
   Type: Dynamic, Manual, Functional
   Initial State: Game started.
   Input/Condition: Testers record players' thoughts about the game display.
   Output/Result: Over 80% of people can recognize game elements clearly.
   How test will be performed: Testing team can invite 10 people to play the game and 8 out of 10 random persons should recognize all the elements clearly.

3. Test-NFR3-LF3
   Type: functional, dynamic, manual
   Initial State: Game started.
   Input/Condition: No inputs will be given for this test case. Players will be asked about their thoughts about the game minimalism.
   Output/Result: Over 80% of players think the game follows the style of minimalism.
   How test will be performed: Invite 10 random persons to play the game and then record their thoughts about the game minimalism.

4. Test-NFR4-LF4
   Type: functional, dynamic, manual
   Initial State: Game started.
   Input/Condition: No inputs will be given for this test case. Players will be asked about their thoughts about the game mood.
   Output/Result: Over 80% of players think the game mood is intense.
   How test will be performed: Invite 10 random persons to play the game and then record their thoughts about the game mood.

### 3.2.2 Usability and Humanity Testing

1. Test-NFR5-UH1
   Type: Dynamic, Manual, Functional
   Initial State: Game started.
   Input/Condition: No inputs will be given for this test case. Child players will be asked about their thoughts of the ease of the game.
   Output/Result: Over 80% of child players think the game is easy.
   How test will be performed: Testing team can invite 10 child players to play the game and record their thoughts about the ease of the game. 8 of 10 children should think the game is easy.

2. Test-NFR6-UH2
   Type: Dynamic, Manual, Functional
   Initial State: Game started.
   Input/Condition: No inputs will be given for this test case. Testing team should record the learning time of players.
   Output/Result: Over 80% of players should be able to play the game with 5 minutes of less learning time.
   How test will be performed: Testing team can invite 10 players to play the game and record their learning time. 8 of 10 players should be able to play the game with 5 minutes or less learning time.

3. Test-NFR7-UH3
   Type: Static, Functional, Manual
   Initial State: Game instructions have been displayed.
   Input/Condition: No inputs will be given for this test case. Testing team should record the time needed for players to understand game rules.
   Output/Result: All players should understand game rules within 10 minutes.
   How test will be performed: Testing team can invite 10 players to read game instructions and record the time needed for players to fully understand the game instructions. All game players should be able to understand game instructions within 10 minutes.

### 3.2.3 Performance Testing

1. Test-NFR8/9-PE1
   Type: Dynamic, Functional, Automated
   Initial state: Game not started.
   Input/Condition: Using external tools to record the response time for each user input.
   Output/Result: All the response time should be less than 1 second.
   How test will be performed: Testing team starts to play the game and tries to pass all game rounds. During this process, testing team should record the response time for each user input, all the response time should be less than 1 second.

2. Test-NFR10-PE2
   Type: Functional, Dynamic, Automated
   Initial State: Game not started.
   Input/Condition: Use an automated program to start the game at random times.
   Output/Result; The game should start successfully each time.
   How test will be performed: Use the automated program to start the game 100 times a day and test like this for 5 days. The game should start properly 500 times in total.

3. Test-NFR11-PE3
   Type: Functional, Dynamic, Manual
   Initial State: Game mode is chosen to be double-player mode.
   Input/Condition: Players start to play the game.
   Output/Result: Games should run properly for a least 2 hours.
   How test will be performed: Testing team can invite 10 players to play in double-player mode. All 5 games should be able to run properly for at least 2 hours.

#### 3.2.4 Operational and Environmental Testing

1. Test-NFR12/13-OE1
   Type: Dynamic, Functional, Automated
   Initial State: Game installed on Windows, Linus and MacOS.
   Input/Condition: Using the automated program to run games on three different platforms.
   Output/Result: Game can run properly over 90% of the time on three different platforms.
   How the test will be performed: Testing team can use the automated program to run the game on three different platforms 100 times. The game should be able to run properly over 90 times.

2. Test-NFR14-OE2
   Type: Functional, Dynamic, Automated
   Initial State: All source codes are implemented.
   Input/Condition: Execute command which can generate `.exe` file for the game.
   Output/Result: `.exe` file can be generated successfully.
   How test will be performed: Testing team can use automated program to execute command which can generate `.exe` file for the game.

3. Test-NFR15-OE3
   Type: Functional, Dynamic, Manual
   Initial State: Game not installed.
   Input/Condition: No inputs will be given for this test case. Testing team will let players to install the game and record the installation process.
   Output/Result: Players should be able to install without any problems.
   How the test will be performed: Testing team can invite 10 people and let them to install the game. All 10 people should not have any problems installing the game.

### 3.2.5 Maintainability and Support Testing

1. Test-NRF16-MS1
   Type: Manual
   Initial state: A new feature is decided to add to the game. (The feature here means the feature may be added after the game is released).
   Input/Condition: Development team starts to prepare MIS and coding. After that, development team tests the newly added feature.
   Output/Result:The process mentioned above (Writing MIS, coding and testing) should be completed within two weeks.
   How test will be performed: After the game is released, project manager should come up with a new feature and let development team to implement this new feature. As a result, we can time how long this new feature can be implemented.

2. Test-NRF17-MS2
   Type: Manual, Dynamic, Automated
   Initial State: Testers open source code and terminal.
   Input/Condition: Testers type command in terminal in order to generate doxygen files.
   Output/Result: Doxygen files should be generated successfully and all contents are correct.
   How test will be performed: Testers can try to generate doxygen files in terminal. After generating all the doxygen files, testers should check if doxygen files contents can match the source code.

3. Test-NFR18-MS3
   Type: Manual
   Initial state: Development team has already read all the previous messages from players.
   Input/Condition: Testing team leaves a message to the development team.
   Output/Result: Testing team should receive a response from the development team.
   How test will be performed: Testing team leaves an advice to the development team in its project repo. After that, testing team will check if they can receive the response from the development team.

4. Test-NFR19-MS4
   Type: Dynamic, Manual
   Initial state: Three computers with three different operating systems(Windows, Linux, MacOS) do not have our game installed.
   Input/Condition: Testing team tries to install our game in three computers.
   Output/Result: Three installations are successful and our game can run properly on three operating systems.
   How test will be performed: Testing team tries to install and run game in three different operating systems and use a checklist to show that game can run properly in three operating systems.

### 3.2.6 Security Testing

1. Test-NFR20-SE1
   Type: Manual, Dynamic
   Initial State: In double-player game mode.
   Input/Condition:Two testers control two aircraft. "Controlling two aircraft" means moving them left and right and shooting bullets from two aircraft.
   Output/Result: A specific aircraft can only be controlled by the corresponding player. For example, aircraft1 should not be controlled by tester2 and aircraft2 should not be controlled by tester1.
   How test will be performed: Two testers start a new game and chooses double-player mode. Tester1 controls aircraft1 and tester2 controls aircraft2. The following three scenarios should be tested:

   - Aircraft1 moves and aircraft2 stays still.
   - Aircraft2 moves and aircraft1 stays still.
   - Two aircraft move concurrently.

   For all three scenarios mentioned above, two aircraft should only be controlled by the correspond tester.

2. Test-NF21-SE2
   Type: Manual, Static
   Initial State: There exists a variable or file(depending on the design decision) to store the highest score.
   Input/Condition: Testing team tries to read from and write to variable/file which stores the highest score.
   Output/Result: Such access should be denied.
   How test will be performed:

   - If the highest score is stored in a variable, testing team can do code inspection to make sure that this variable is private and there are no getter and setter for this variable.
   - If the highest score is stored in a file, this file should display "encrypted" when tester tries to access it.

### 3.2.7 Cultural and Political Testing

1. Test-NFR22-CP1
   Type: Manual, Dynamic
   Initial state: Game not started.
   Input/Condition: Testing team starts to play game and record the process of playing.
   Output/Result: There should not be any offensive messages or images in the recording.
   How test will be performed: Testing team starts to play the game and record the screen. After that, testing team will analysis the recording to ensure that there are no offensive messages or pictures.

### 3.2.8 Legal Testing

1. Test-NF23-LE1 Type: Manual, Dynamic
   Initial state: Game not started.
   Input/Condition: Testing team starts to play game and record the process of playing.
   Output/Result: There should not be any illegal things in the recording.
   How test will be performed: Testing team starts to play the game and record the screen. After that, testing team will analysis the recording to ensure that everything is legal. Maybe testing team can invite domain experts.

### 3.2.9 Health and Safety Testing

1. Test-NF24-HS1 Type: Functional, Dynamic, Manual
   Initial state: Game not started.
   Input/Condition: Testers start the game.
   Output/Result: GUI shows a message to notify the player to avoid game addiction with the welcoming message.
   How test will be performed: Testing team should run the program and make sure that game addiction avoidance message is showed with the welcoming message.

## 3.3 Traceability Between Test Cases and Requirements

### 3.3.1 Model Traceability Matrices

Table 4: Model Traceability Matrix 1

| Tests/Requirement | FR15 | FR16 | FR25 | FR26 | FR27 | FR29 |
|---|---|---|---|---|---|---|
| Test-FR15-M1 | X | | | | | |
| Test-FR16-M2 | | X | | | | |
| Test-FR16-M3 | | X | | | | |
| Test-FR25-M4 | | | X | | | |
| Test-FR26-M5 | | | | X | | |
| Test-FR27-M6 | | | | | X | |
| Test-FR29-M7 | | | | | | X |

Table 5: Model Traceability Matrix 2

| Tests/Requirement | FR30 | FR31 | FR32 | FR33 | FR34 | FR35.1 | FR39 |
|---|---|---|---|---|---|---|---|
| Test-FR30-M8 | X | | | | | | |
| Test-FR31-M9 | | X | | | | | |
| Test-FR32-M10 | | | X | | | | |
| Test-FR33-M11 | | | | X | | | |
| Test-FR34-M12 | | | | | X | | |
| Test-FR35.1-M13 | | | | | | X | |
| Test-FR39-M14 | | | | | | | X |

### 3.3.2 View Traceability Matrices

Table 6: View Traceability Matrix 1

| Tests/Requirement | FR1 | FR2 | FR3 | FR4 | FR5 | FR6 | FR7 | FR8 | FR9 |
|---|---|---|---|---|---|---|---|---|---|
| Test-FR1-V1 | X | | | | | | | | |
| Test-FR2-V2 | | X | | | | | | | |
| Test-FR3-V3 | | | X | | | | | | |
| Test-FR4-V4 | | | | X | | | | | |
| Test-FR5-V5 | | | | | X | | | | |
| Test-FR6-V6 | | | | | | X | | | |
| Test-FR7-V7 | | | | | | | X | | |
| Test-FR8-V8 | | | | | | | | X | |

Table 7: View Traceability Matrix 2

| TestCase\Requirements | FR9 | FR10 | FR11 | FR12 | FR13 | FP14 | FR28 |
|---|---|---|---|---|---|---|---|
| Test-FR9-V9 | X | | | | | | |
| Test-FR9-V10 | X | | | | | | |
| Test-FR10-V11 | | X | | | | | |
| Test-FR11-V12 | | | X | | | | |
| Test-FR12-V13 | | | | X | | | |
| Test-FR13-V14 | | | | | X | | |
| Test-FR14-V15 | | | | | | X | |
| Test-FR28-V16 | | | | | | | X |
| Test-FR28-V17 | | | | | | | X |
| Test-FR28-V18 | | | | | | | X |

### 3.3.3 Control Traceability Matrices

Table 8: Control Traceability Matrix 1

| Tests/Requirement | FR17 | FR18 | FR19 | FR20 | FR21 | FR22 |
|---|---|---|---|---|---|---|
| Test-FR17-C1 | X | | | | | |
| Test-FR18-C2 | | X | | | | |
| Test-FR19-C3 | | | X | | | |
| Test-FR20-C4 | | | | X | | |
| Test-FR21-C5 | | | | | X | |
| Test-FR22-C6 | | | | | | X |

Table 9: Control Traceability Matrix 2

| Tests/Requirement | FR23 | FR24 | FR35.2 | FR36 | FR37 | FR38 |
|---|---|---|---|---|---|---|
| Test-FR23-C7 | X | | | | | |
| Test-FR24-C8 | | X | | | | |
| Test-FR35.2-C9 | | | X | | | |
| Test-FR36-C10 | | | | X | | |
| Test-FR37-C11 | | | | | X | |
| Test-FR38-C12 | | | | | | X |

### 3.3.4 Nonfunctional Req Test Matrices

Table 10: Nonfunctional Req Test Matrix 1

| Tests/Requirement | NFR1 | NFR2 | NFR3 | NFR4 | NFR5 | NFR6 | NFR7 |
|---|---|---|---|---|---|---|---|
| Test-NFR1-LF1 | X | | | | | | |
| Test-NFR2-LF2 | | X | | | | | |
| Test-NFR3-LF3 | | | X | | | | |
| Test-NFR4-LF4 | | | | X | | | |
| Test-NFR5-UH1 | | | | | X | | |
| Test-NFR6-UH2 | | | | | | X | |
| Test-NFR7-UH3 | | | | | | | X |

Table 11: Nonfunctional Req Test Matrix 2

| Tests/Requirement | NFR8 | NFR9 | NFR10 | NFR11 | NFR12 | NFR13 | NFR14 | NFR15 | NFR16 |
|---|---|---|---|---|---|---|---|---|---|
| Test-NFR8/9-PE1 | X | X | | | | | | | |
| Test-NFR10-PE2 | | | X | | | | | | |
| Test-NFR11-PE3 | | | | X | | | | | |
| Test-NFR12/13-OE1 | | | | | X | X | | | |
| Test-NFR14-OE2 | | | | | | | X | | |
| Test-NFR15-OE3 | | | | | | | | X | |
| Test-NFR16-MS1 | | | | | | | | | X |

Table 12: Nonfunctional Req Test Matrix 3

| Tests/Requirement | NFR17 | NFR18 | NFR19 | NFR20 | NFR21 | NFR22 | NFR23 | NFR24 |
|---|---|---|---|---|---|---|---|---|
| Test-NFR17-MS2 | X | | | | | | | |
| Test-NFR18-MS3 | | X | | | | | | |
| Test-NFR19-MS4 | | | X | | | | | |
| Test-NFR20-SE1 | | | | X | | | | |
| Test-NFR21-SE2 | | | | | X | | | |
| Test-NFR22-CP1 | | | | | | X | | |
| Test-NFR23-LE1 | | | | | | | X | |
| Test-NFR24-HS1 | | | | | | | | X |

# 4 Tests for Proof of Concept

For the POC demo, we only implemented one aircraft and three monsters with three different colors. As a result, our testing areas will be Aircraft and Monster.

## 4.1 Aircraft Tests

1. Test1-POC
   Type: Functional, Dynamic, Manual
   Initial State: One aircraft is displayed on the screen.
   Input/Condition: Players press ← key.
   Output/Result: Aircraft moves to the left.
   How the test will be performed: Developers start the game by executing `python Driver.py` and then press ← to check if aircraft can move left.

2. Test2-POC Type: Functional, Dynamic, Manual
   Initial State: One aircraft is displayed on the screen.
   Input/Condition: Players press → key.
   Output/Result: Aircraft moves to the right.
   How the test will be performed: Developers start the game by executing `python Driver.py` and then press → to check if aircraft can move right.

3. Test3-POC
   Type: Functional, Manual, Dynamic
   Initial State: One aircraft is displayed on the screen.
   Input/Condition: Players press `SPACE` key.
   Output/Result: Aircraft shoots one bullet.
   How the test will be performed: Developers start the game by executing `python Driver.py` and then press `SPACE` to check if aircraft can shoot one bullet.

## 4.2 Monster Tests

1. Test-POC4
   Type: Functional, Dynamic, Manual
   Initial State: One green monster is displayed on the screen.
   Input/Condition: Shoot one bullet to the green monster.
   Output/Result: Green monster disappears.
   How the test will be performed: Developers start the game by executing `python Driver.py` and then shoot one bullet the green monster.

2. Test-POC5
   Type: Functional, Dynamic, Manual
   Initial State: One blue monster is displayed on the screen.
   Input/Condition: Shoot two bullets to the blue monster.
   Output/Result: Blue monster disappears.
   How the test will be performed: Developers start the game by executing `python Driver.py` and then shoot two bullets the blue monster.

3. Test-POC6
   Type: Functional, Dynamic, Manual
   Initial State: One pink monster is displayed on the screen.
   Input/Condition: Shoot three bullets to the pink monster.
   Output/Result: Pink monster disappears.
   How the test will be performed: Developers start the game by executing `python Driver.py` and then shoot three bullets the pink monster.

# 5 Comparison to Existing Implementation

When this document is finished, the following contents have been implemented:

- Monster Model
- SpaecShip Model
- Score Model
- Bullet Model
- Single Player Controller
- Background, Title and Icon
- The corresponding views of the four models

# 6 Unit Testing Plan

## 6.1 Unit testing of internal functions

Unit tests for this project will mainly be applied to test our models. The unit testing frame we choose is `Pytest`. The following are steps for of unit testing:

1. Create object from different model class.
2. Provide different inputs and call different methods.
3. Compare the return values with the excepted values.

Some sample examples for unit testing:

- Monster Class
  - `reduce_life()`
  - `isDead()`
  - setter and getters for x, y coordinates.
- Score Class
  - `increaseScore()`
  - `getScore()`

## 6.2 Unit testing of output files

This game will not generate and output files. As a result, no unit tests will be applicable for output files.

# 7 Appendix

## 7.1 Symbolic Parameters

- `DEFAULT_SPACESHIP_SPEED = 3`
- `DEFAULT_SPACESHIP_X = 370`
- `DEFAULT_SPACESHIP_Y = 530`
- `DEFAULT_MONSTER_SPEED = 1`
- `DEFAULT_BULLET_SPEED = 20`

## 7.2 Usability Survey Questions

- How would you describe the difficulty of this game? (Level 10 is the most difficult and level 1 is the least difficult)
- Do you have any recommended features that we should implement in the future?
- Is this game difficult to learn to play? (Level 10 is the most difficult and level 1 is the least difficult)