****

**Group 4**

**Rachelle Runyon**

**Seda Alioglu**

**Bartosz Jelen**

**TABLE OF CONTENTS**

**Part 1:**

Problem Statement……………………………………………………………………...……...………..5

Glossary of Terms………………………………………………………………………………………..6

**Part 2: Appendix A**

A. Requirements Strategy……………………………………………………………………………….7

A.1 User Requirements Specification……………………………………………………………..……7

A.1.1 User Requirements………………………………………………………………….……8

A.2 System Requirements Specification………………………………………………...……………10

A.2.1 Enumerated Functional Requirements…………………………………………..……10

A.2.2 Enumerated Non-Functional Requirements……………………………….…………14

A.3 Functional Requirements Specification……………………………..……………………………16

A.4 Traceability Matrix……………………………………………………………………….…………18

**Part 3- Appendix B**

B.1 Use Case Diagram…………………………………………………………………………………19

B.2 Sequence Diagrams for UC 1-8……………………………………………………………..……20

B.2.1 Sequence Diagrams UC 1A……………………………………………………………21

B.2.2 Sequence Diagrams UC 1B……………………………………………………………22

B.2.3 Sequence Diagrams UC 2………………………………………………...……………23

B.2.4 Sequence Diagrams UC 3…………………………………………………………...…24

B.2.5 Sequence Diagrams UC 4……………………………………………………...………25

B.2.6 Sequence Diagrams UC 5……………………………………………………………...26

B.2.7 Sequence Diagrams UC 6……………………………………………………………...27

B.2.8 Sequence Diagrams UC 7……………………………………………………………...28

B.2.9 Sequence Diagrams UC 8……………………………………………………………...29

B.2.10 Sequence Diagrams UC 9…………………………………………………...……….30

B.3.1 Class Diagram……………………………………………………………………………………31

B.4 Architectural Diagrams…………………………………………………………………………….32

B.4.1 Activity Diagram…………………………………………………………………………33

B.4.2 Client-Server Diagram………………………………………………………………….34

B.4.3 Layered Diagram………………………………………………………………………..35

B.5 Test Cases………………………………………………………………………………………….36

B.5.1 Test Case UC1A………………………………………………………………………...36

B.5.2 Test Case UC1B…………………………………………………………………………37

B.5.3 Test Case UC2…………………………………………………………………………..37

B.5.4 Test Case UC3 ………………………………………………………………………….38

B.5.5 Test Case UC4………………………………………………………………………..…39

B.5.6 Test Case UC5………………………………………………………………………..…40

B.5.7 Test Case UC7……………………………………………………………..……………41

B.5.8 Test Case UC8 ……………………………………………………………….…………41

**Part 4 - Appendix C**

C.1 Game Visuals…………………………………………………………….…………………………42

C.1.1 Screenshots………………………………………………………...……………………42

C.1.2 Screen Recording………………………………………………………………….……43

**Part 5 - Appendix D**

D.1 Source code……………………………………………………………………………...…………44

D.1.1: Source Code: Arrow……………………………………………………………………44

D.1.2: Source Code: Bullet………………………………………………………………….…45

D.1.3: Source Code: Button………………………………………………………………...…46

D.1.4: Source Code: game\_functions……………………………………………..…………46

D.1.5: Source Code: game\_stats………………………………………………………..……50

D.1.6: Source Code: monster…………………………………………………………………51

D.1.7: Source Code: monster\_invasion……………………………………...………………52

D.1.8: Source Code: scoreboard……………………………………………………..………53

D.1.9: Source Code: settings…………………………………………………………….……55

**Part 6 - Appendix E**

E.1 References…………………………………………………………………………………….……56

**Table of contents for Tables:**

Table A.1.1: User Requirements for “Monster Invasion” Game……………..………………………8

Table A.2.1: Enumerated Functional Requirements…………………………...……………………10

Table A.2.2: Enumeration Non-Functional Requirements…………………….……………………14

Table A.3.1: Traceability matrix mapping along with priority weight (PW)…..……………………18

Table B.5.1: Test for Use Case 1A…………………………...………………….……………………36

Table B.5.2: Test for Use Case 1B…………………………...………………….……………………37

Table B.5.3: Test for Use Case 2……………………………...………………………………………37

Table B.5.4: Test for Use Case 3……………………………...………………………………………38

Table B.5.5: Test for Use Case 4……………………………...………………………………………39

Table B.5.6: Test for Use Case 5……………………………...………………………………………40

Table B.5.7: Test for Use Case 7……………………………...………………………………………41

Table B.5.8: Test for Use Case 8……………………………...………………………………………41

**Table of contents for Figures:**

Figure A.1: Requirements development strategy………………………..……………………………7

Figure B.1.1: Use case diagram……………………………………………….………………………20

Figure B.2.1: Sequence Diagram - Use Case 1A……………………………………………………21

Figure B.2.2: Sequence Diagram - Use Case 1B……………………………………………………22

Figure B.2.3: Sequence Diagram - Use Case 2……………………………………..………………23

Figure B.2.4: Sequence Diagram - Use Case 3………..……………………………………………24

Figure B.2.5: Sequence Diagram - Use Case 4…………..…………………………………………25

Figure B.2.6: Sequence Diagram - Use Case 5…………..…………………………………………26

Figure B.2.7: Sequence Diagram - Use Case 6…………..…………………………………………27

Figure B.2.8: Sequence Diagram - Use Case 7…………..…………………………………………28

Figure B.2.9: Sequence Diagram - Use Case 8 ………….…………………………………………29

Figure B.2.10: Sequence Diagram - Use Case 9 ………...…………………………………………30

Figure B.3.1: Class Diagram………………………………...…………………………………………31

Figure B.4.1: Activity Diagram ……………………………...…………………………………………33

Figure B.4.2: Client-Server Diagram …………………………………………………………………34

Figure B.4.3: Layered Diagram ………………………………….……………………………………35

**Part 1: Customer Problem Statement**

The basic model of game “Monster Invasion” requires a number of updates concerning its functionality and security. By introducing an updated version of the video game, we are expecting to solve pre-existing bugs existent from the first version of the game, as well as make it more user friendly and profitable.

The main functional improvements we consider are: adding limited number of bullets and lives, improving sound effects, and keeping track of user’s highest scores. Other than functional problems, the first version of the game did not offer any game modes. However, the new version of the game is expected to introduce multiplayer mode and allow for communication between users. At the final stage of the game, the system will allow for payments which would omit advertisements, purchase more lives, and purchase more bullets. It will also Improve security by providing secure payments and solving existing vulnerabilities. This is expected to be the final and crucial improvement of the game.

**Glossary of Terms**

* “Monster Invasion”- Name of the video game.
* Slime bomb- The “bomb” that shoots out of the slime when the spacebar is pressed to kill the falling monsters.
* Play screen- Initial game screen where user will press play to begin the game.
* Leaderboard- a scoreboard displaying the names and current scores of the leading competitors.
* Game Character- refers to the monsters that are to be shot at by the slime bomb. These characters can be upgraded by the user if they choose to purchase a different character.

**PART 2- APPENDIX A**

**A: REQUIREMENTS - STRATEGY**

Considering we are using a hybrid of reused-based process model combined with agile type of incremental process for developing our game, we decided that two separate requirements are necessary- system and user requirements.

User requirements should be based on discussion with the client. Client’s preferences, needs and expectation should be considered and presented to the developing department. User requirements should describe main functionalities of our updated game (what the client/user wants the product to do). Based on that, system requirements should be developed (functional + non functional). The first version of the game can be created and developed with the use of old components of the old game, but according to new requirements. Every version of the game will be updated after completing set of 3 requirements. This technique will allow to detect bugs and changes at early stage. The first version should be analyzed with the client and necessary changes in requirements should be made (user opinion and experiences will be taken into consideration). The time requirements need to be very specific and final. The client should approve this requirements list. The system requirements should also be updated after client’s approval for user requirements. Final requirements are expected to be complete in 4 weeks. The following chart presents requirements developing strategy.

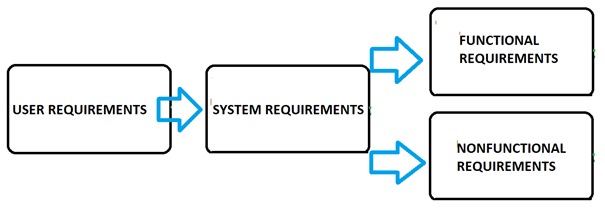
**

FIGURE A.1: Requirements development strategy

**A.1: User Requirements**

User requirements are states, in a natural language accompanied by diagrams, of what services the system is expected to provide to system users and the constraints under which the system must operate. After meeting with our client, our customer explained that the project is expected to be completed in 12 weeks. He also provided the following user requirements (Top 3 requirements are expected in first version of the game):

**Table A.1.1: User Requirements for "Monster Invasion" Game**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Identifier*** | ***Requirement*** | ***TIME*** | ***PW*** | ***VERSION*** |
| ***REQ01*** | *Develop 100% functional free version of game, on the marker* | *Week 1-3* | *3* | *v1.0* |
| ***REQ02*** | *System should allow limited number of bullets (after in user has to wait)* | *Week 1-3* | *1* | *v1.0* |
| ***REQ03*** | *System should allow for limited number of life (after in user has to wait)* | *Week 1-3* | *1* | *v1.0* |
| ***REQ04*** | *System should allow to play online / different modes of the game* | *Week 3-6* | *3* | *v2.0* |
| ***REQ05*** | *System should allow to request games with other players online* | *Week 3-6* | *2* | *v2.0* |
| ***REQ06*** | *System should allow for sound effects in the game* | *Week 3-6* | *1* | *v2.0* |
| ***REQ07*** | *System should allow for ads (watching adds can be later used instead of payment)* | *Week 7-10* | *3* | *v3.0* |
| ***REQ08*** | *System should allow for online payments (instead of adds)* | *Week 7-10* | *3* | *v3.0* |
| ***REQ09*** | *System should allow for communication between users* | *Week 7-10* | *1* | *v3.0* |
| ***REQ10*** | *System should allow to keep track of highest scores* | *Week 10-12* | *4* | *V4.0* |
| ***REQ11*** | *System should allow for different background sounds* | *Week 10-12* | *2* | *V4.0* |
| ***REQ12*** | *System should display number of lives that user has left* | *Week 10-12* | *1* | *V4.0* |

The main requirement (REQ01) is to get the fully functional game into the market as fast as possible. This step is especially important in incremental model, in which next version of the game allows to fix bugs from previous version of it. The second and third requirements (RQ02 and RQ03) require system to offer life and bullet limitation. After certain number of bullets and life’s, the user will have to wait a certain amount of time (for details go to system req). After completing first three requirements, the first version of the game should be deployed to the market for free. Moreover, there should be protocol created, that will keep track of changes in the game as well as fixed bugs.

The second version of the game will take care of REQ 4-6. In this version game will be updated by: RQ04 - possibility to play online (multiplayer mode), RQ05 - requesting other online players to share game (players should allow to accept or refuse the challenge), RQ06 sound effects (add on/ off button). Also, bugs from version one need to be fixed at this point. Next, we deploy 2nd version of the game that is still free to the user.

In the third version of the game, we are planning to add ads – RQ7. While waiting for life or bullets user will have to watch short advertisement, similar to those on YouTube. Following requirement - RQ08 should allow user for online payments instead of waiting. Third version also allows users to communicate with each other – RQ09. After completing requirements 7-9 all bugs from version two should be fixed. Next, version number three is expected to be deploy during week 10.

Final version is expected to be on the market after completing REQ 10 -12. RQ10, adds table with the highest scores. RQ11, allows user to choose background sounds for his game. RQ12, during game, user should be available to check how many life has left.

**A.2: System Requirements**

System requirements are more detailed descriptions of the software system’s functions, services, and operational constraints. The system requirements defines exactly what is to be implemented. The system requirements are often one of two requirements: functional requirements or non-functional requirements.

Functional requirements are statements that provide insight on how the system should react to certain inputs and how the system should behave in certain situations. Sometimes the functional requirements also specifically states what the system should not do as well.

Non-functional requirements are constraints on the services or functions provided by the system. These often refer to the system as a whole, rather than individual system features or services. Non-functional requirements may affect the overall architecture of a system rather than the individual components. Sometimes one non-functional requirement may lead to new functional requirements that define new system services that are required. Some non-functional requirements may also lead to the restriction of existing requirements.

**Table A.2.1: ENUMERATED FUNCTIONAL REQUIREMENTS**

\*\*PW scale 1-5 (1 being low importance and 5 being high importance)

|  |  |  |
| --- | --- | --- |
| IDENTIFIER | REQUIREMENT DESCRIPTION | PW |
| REQ 1 | The system shall allow the user to register with a specific username of their choice. | 3 |
| REQ 2 | The system shall give an error message when there is a duplicate username entry during registration. | 3 |
| REQ 3 | The system shall offer the user the option to play against another person. | 5 |
| REQ 3a | The system shall offer the user the option to enter another user’s username to play against them or choose a random opponent. | 4 |
| REQ 3b | The system shall pick the random opponent from the database of users who are also seeking to play against a random opponent. | 4 |
| REQ 4 | The system shall allow each player to only play one game at a time. | 3 |
| REQ 5 | The system shall allow each player the opportunity to forfeit and end a match while playing with an opponent. | 3 |
| REQ 6 | The system will end every match in either a win or a draw, if multiple users are playing. | 4 |
| REQ 7 | The system shall not allow the users to pause the game while in multiplayer mode. | 2 |
| REQ 8 | The system shall allow users to navigate the slime bomb across the screen with the left and right arrow keys. | 1 |
| REQ 9 | The system shall allow users to shoot bullets with the space bar. | 1 |
| REQ 10 | The system shall make a sound with every bullet that is shot. | 1 |
| REQ 11 | The system shall increase the overall score by 50 every time a monster is hit with a bullet. | 2 |
| REQ 12 | The system shall delete the monster when it is hit by the bullet. | 2 |
| REQ 13 | The system shall take a life away from the user once the monster reaches the bottom of the screen. | 4 |
| REQ 14 | The system shall provide the user with four free lives when they begin the game. | 3 |
| REQ 15 | The system shall allow the user to quit the application by pressing the ‘q’ button. | 2 |
| REQ 16 | The system shall allow the user to pause the game using the ‘p’ button. | 2 |
| REQ 17 | The system shall offer upgrades every 10 levels. | 3 |
| REQ 18 | The system shall end the game once user uses up all of their lives. | 4 |
| REQ 19 | The system shall start a 15 minute timer [to load another life] as soon as the user loses their first life. | 4 |
| REQ 20 | The system shall display the leaderboard and a play again button once all four lives are lost and the level is not passed. | 3 |
| REQ 21 | The system shall not restart the user’s score after every life is lost, until the user loses all four of their lives. | 4 |
| REQ 22 | The system shall offer the user an option to purchase more lives or bullets. | 5 |

**System Requirement Descriptions- Functional Requirements**

\*\*Note: REQ descriptions are only for the requirements that will not be implemented into the game.

* The requirement (REQ3) specifies that any player can invite any other player or only the players who are not already playing. We may introduce an option that the system informs the inviter that they must wait until the invitee completes the ongoing game first. This issue is related to (REQ38).
* The requirement (REQ19) specifies that the system shall start a 15 minute timer as soon as the user loses their first life, to start reloading their new lives. Depending on how fast or slow the user completes each level without losing a life, the user may or may not have any wait for lives to be restarted.This issue is related to (REQ34).
* The requirement (REQ22) specifies that there is an option to purchase more lives and bullets. If the user would like to purchase more bullets, they will have the option to add either 15, 20, or 25 bullets per shot instead of the original 10 bullets per shot. If the user would like to purchase more lives, they will have the option to buy a minimum of five for a certain price, up to an unlimited amount of lives for a certain price. This requirement adds the aspect of security in the game. The user will have to create an account within the game and add their credit card information to finalize the purchase. Because personal credit card information, it is very important that the program is a safe and secure network. This issue is related to (REQ32).
* The requirement (REQ21) specifies that the system will show a leaderboard of leading competitors and their ranking.The ranking is based off of each users highest level obtained. A user who chooses to purchase extra lives will get more opportunities to beat the levels, but the game will start over when all lives are lost and the user will again have to start at level one. The score the user finishes with when all the lives are gone is what will go towards the leaderboard. For each level, only the top 20 scores from users will make it onto the user board. This issue is related to (REQ33). If a user pauses the game before they begin the next level, they can start from that point next time they play, in single player mode only. This issue is related to (REQ29).The system has a database in which all the scores will be stored. This issue is related to (REQ26).
* The requirement (REQ17) specifies that the system will offer upgrades every 10 levels. This means, on level 10, 20, etc… the user will have the option to add certain updates like: changing the color of the slime bomb, choosing a different monster to shoot, or changing the style of the bullet. If a user gets to level 20 then loses level on level 21, the game is reset back to zero but the user will still keep that upgrade. It will not be until they reach level 20 again that they can update the choice of their upgrade from the last time they were at level 20.This issue is related (REQ28).
* The requirement (REQ4) specifies that’s the system will allow each user to only play one game at a time. Because the user has to set up an account to play, the system will not allow the same user to be active in more than one game at a time. This will stop users from getting around purchasing more lives to move up the leaderboard and cheat the system.
* The requirement (REQ5) specifies that a player can’t forfeit in the middle the match while playing against another user. If this case happens, the other opponent will automatically win and be able to continue from that level in future competitions.
* The requirement (REQ1) specifies that the user can register with any username. To play the game, the user must create an account where they create a username that will be used for the leaderboard and when other users search to play against them. The username must be at least 10 characters long. The username will be started in the systems database. This issue is related to (REQ24) and (REQ26).
* The requirement (REQ6) specifies that the game will end in either a win or a draw of multi-players are playing against each other. The game does not allow for a tie because if the score is high enough to make it into the leaderboard, only 20 names and 20 different scores can be printed.
* The requirement (REQ2) specifies that an error message will be displayed if a user tries to create an account with a username that is already taken. The system cannot have duplicate names because the leaderboard cannot have more than one user displayed with the same name. The user will have unlimited attempts to change their username until they find one that is original.
* The requirement (REQ7) specifies that a user can not pause the game while in multiplayer mode so both players start and finish at the same time.

**Table A.2.2: ENUMERATED NON-FUNCTIONAL REQUIREMENTS**

|  |  |
| --- | --- |
| IDENTIFIER | REQUIREMENT DESCRIPTION |
| REQ 23 | The system shall have algorithms determining the speed of the bullets. |
| REQ 24 | The system shall restrict the username input to only 10 characters. |
| REQ 25 | The system shall have a database where everything is stored. |
| REQ 26 | The system shall store all the usernames in the database along with their highest level achieved. |
| REQ 27 | The system shall replace the old stored level with the newest level achieved by the same user. |
| REQ 28 | The system shall retrieve the top 20 highest levels achieved [with usernames] to display on the leaderboard. |
| REQ 29 | The system shall store every users latest achieved level, so the user can pick up where they left off. |
| REQ 30 | The system shall have algorithms determining the speed at which the slime bomb moves across the screen. |
| REQ 31 | The system shall pick random players from the database. |
| REQ 32 | The system shall have a secure network for any in-app purchases. |
| REQ 33 | The system shall be storing all of the scores and rank them in a descending order if the user makes it into the top 20 in the leaderboard. |
| REQ 34 | The system shall have a built in timer to start when players lose their lives. |
| REQ 35 | The system shall require minimum maintenance. |
| REQ 36 | The system should respond quickly to users’ operations (no wait time after clicking or selecting a button). |
| REQ 37 | The system shall shoot 10 bullets per 1 space bar shot. |
| REQ 38 | The system shall allow the user to invite another player to compete against them in the game. |

**System Requirement Descriptions- Non Functional Requirements**

* The requirement (REQ27) states how the system will only store the highest score achieved by the user in the database. Once the user beats their high score, that score will remain in the system until it is beat.
* The requirement (REQ31) states how the system will pick random players from the database if a player wants to play in multiplayer mode, but does not have a specific player’s username they want to play against.
* The requirement (REQ25) states how the system will have a database where all of the user input will be stored. This information includes the user’s username, the level they left off at, their position on the leaderboard, along with payment information. (The payment information is being stored to make it easier to make future purchases in the application.)

**A.3: FUNCTIONAL REQUIREMENTS SPECIFICATION**

***Stakeholders***

* User
* Administrator

***Actors and Goals***

* Player 1
  + Initiating actor
  + Player 1 has the ability to open the game, register and play it in the mode of their choice (single or multiplayer)
* Player 2
  + Participating actor
  + The goal of the player 2 is to be the challenger of player 1.
* Application
  + Participating actor
  + The goal of the application is to provide a fun source of entertainment. The application is run on a web browser.
* Visitor
  + An unregistered user
  + The visitor has the ability to visit the game and view the leaderboard.
* Database
  + Participating actor
  + The goal of the database is to collect and store every player’s name and highest score, along with whatever purchases the player may have made.

***Use Case Descriptions***

* UC-1: Register User
  + Allows the player to register themselves and become a member of the game. Registering themselves lets the user hold a position on the leaderboard, as well as, remembers which level the user left off. This way the user’s progress throughout the game is not lost. (REQ 1-2)
* UC-2: Challenge
  + Allows the player to challenge an opponent by entering their username in the field. (REQ 3a)
    - Extension point: The chosen opponent has the option to decline the game.
* UC-3: Multiplayer Mode
  + Allows the user to play against an opponent of either their choice (by using the username of their opponent) or a random opponent picked from the database. They players can only play one game at a time. Each player will have the option to forfeit and a matching while playing. The players will not have the option to pause the game once it starts. Every match is to end with either a win or a draw. A player wins once they are the first one to kill all of the monsters. (REQ 3-7)
* UC-4: Playing Game
  + The player will be able to navigate the slime bomb across the screen with the right and left arrows and will be able to shoot bullets with the space bar. For every monster the player shoots they get 50 points. The monster that gets hit disappears.Each user gets four free lives when the game begins. The players lose a life when the monsters reach the bottom of the screen. New lives reappear every 15 minutes after each life is lost. Players can pause the game by pressing the ‘p’ button or quitting the game by pressing the ‘q’ button. When player kills all monsters without losing all four lives they progress to the next level. Every 10 levels, the players are offered character upgrades. (REQ 8-19)
* UC-5: Lost game
  + When the player loses all their lives and cannot pass the level they will be directed to see the top 20 players who made the leaderboard along with their highest level achieved and the option to play again, once their lives reload [or they purchase more lives] (REQ 20-21)
* UC-6: Purchases
  + The players will always have the option to purchase more lives or bullets that shoot at a faster pace. (REQ 22)
    - Extension point: If the player is not a registered player they will have to input their payment information if they wish to purchase anything.
* UC-7: Viewing leaderboard
  + The player will have the option to view the leaderboard. They will be able to see the top 20 players with the highest levels attained [based off all the data stored in the database]. (REQ 20a)
* UC-8: Game Mode
  + The player will be able to choose between playing in single player mode or playing in multiplayer mode. (REQ 3-3b)
* UC-9: Single player mode
  + The player will be able to play the game by themselves. (everything from use case-4 is applicable here). (REQ 8-19)

**A.4 Traceability Matrix**

Traceability policies define the relationship between the requirements and the system design. The traceability matrix in table 4 shows how our system requirements map to our use cases. We calculated the priority weights of the use cases, and we can order our use cases by priority:

UC-3, UC-4, UC-9 > UC-8 > UC-2 > UC-5 > UC-1 > UC-6 > UC-7

**Table A.3.1: Traceability matrix along with priority weight (PW)**

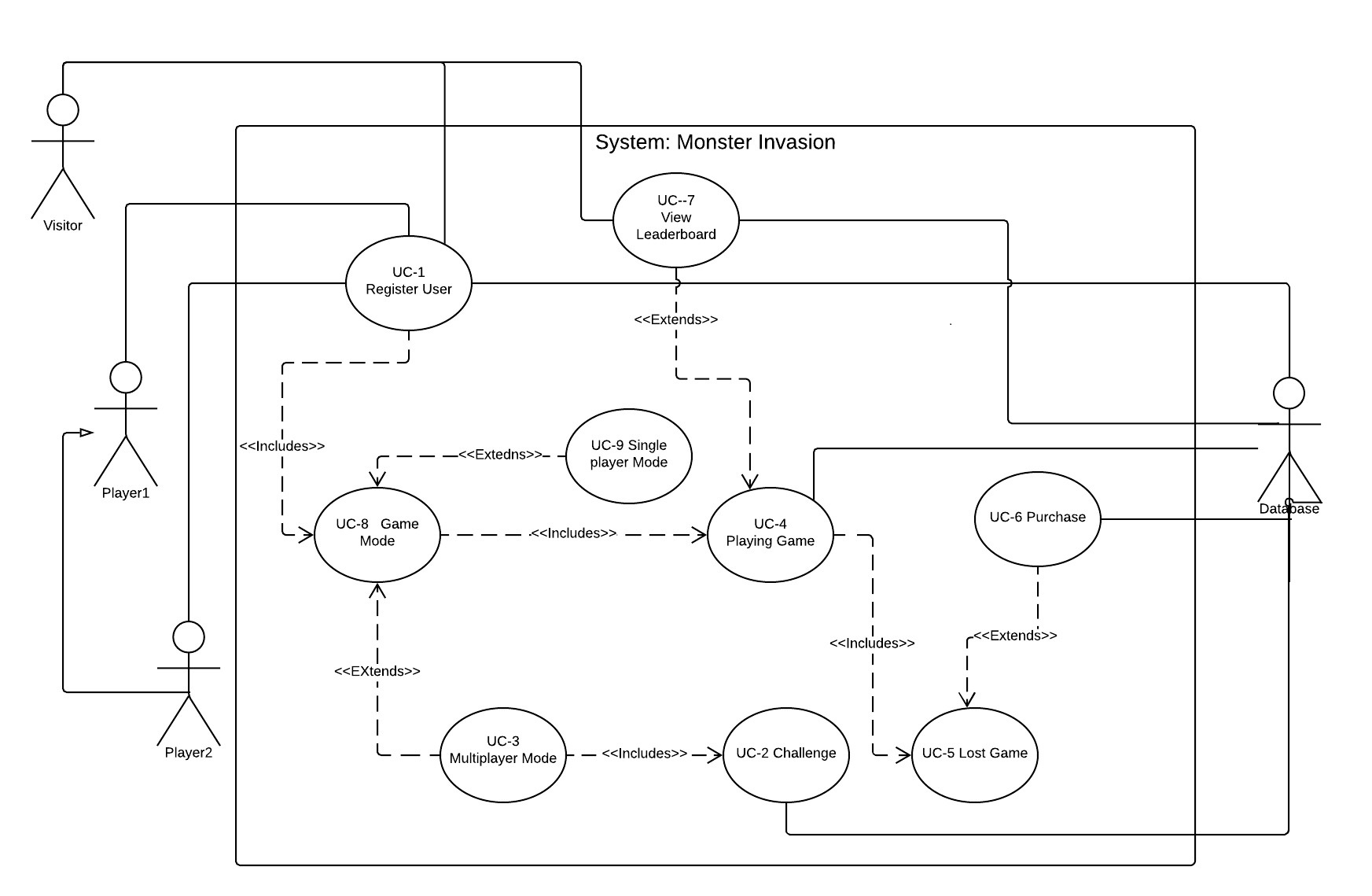
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REQ | PW | UC-1 | UC-2 | UC-3 | UC-4 | UC-5 | UC-6 | UC-7 | UC-8 | UC-9 |
| REQ 1 | 3 | X |  |  |  |  |  |  |  |  |
| REQ 2 | 3 | X |  |  |  |  |  |  |  |  |
| REQ 3 | 5 |  | X | X |  |  |  |  | X |  |
| REQ 3a | 4 |  | X | X |  |  |  |  | X |  |
| REQ 3b | 4 |  |  | X |  |  |  |  | X |  |
| REQ 3c | 4 |  |  | X |  |  |  |  |  |  |
| REQ 4 | 3 |  |  | X |  |  |  |  |  |  |
| REQ 5 | 3 |  |  | X |  |  |  |  |  |  |
| REQ 6 | 4 |  |  | X |  |  |  |  |  |  |
| REQ 7 | 2 |  |  | X |  |  |  |  |  |  |
| REQ 8 | 1 |  |  |  | X |  |  |  |  | X |
| REQ 9 | 1 |  |  |  | X |  |  |  |  | X |
| REQ 10 | 1 |  |  |  | X |  |  |  |  | X |
| REQ 11 | 2 |  |  |  | X |  |  |  |  | X |
| REQ 12 | 2 |  |  |  | X |  |  |  |  | X |
| REQ 13 | 4 |  |  |  | X |  |  |  |  | X |
| REQ 14 | 3 |  |  |  | X |  |  |  |  | X |
| REQ 15 | 2 |  |  |  | X |  |  |  |  | X |
| REQ 16 | 2 |  |  |  | X |  |  |  |  | X |
| REQ 17 | 3 |  |  |  | X |  |  |  |  | X |
| REQ 18 | 4 |  |  |  | X |  |  |  |  | X |
| REQ 19 | 4 |  |  |  | X |  |  |  |  | X |
| REQ 20 | 3 |  |  |  |  | X |  |  |  |  |
| REQ 20a | 3 |  |  |  |  |  |  | X |  |  |
| REQ 21 | 4 |  |  |  |  | X |  |  |  |  |
| REQ 22 | 5 |  |  |  |  |  | X |  |  |  |
| Max PW | | 3 | 5 | 5 | 4 | 4 | 5 | 3 | 5 | 4 |
| Total PW | | 6 | 9 | 29 | 29 | 7 | 5 | 3 | 13 | 29 |

**PART 3- APPENDIX B**

**B.1: Use Case Diagram for the distributed game of “Monster Invasion”**

A use case diagram is a dynamic or behavior diagram in UML. They model the functionality of a system using actors and use cases. The use cases are a set of actions, services, and functions that the system needs to perform. A “system” is something that is being developed or operated and an “actor” is the person or entity using the system.

Use case diagrams are valuable for visualizing the functional requirements of a system. They also help identify any internal or external factors that may influence the system and should be taken into consideration. Use case diagrams provide a good high level analysis from outside the system. They specify how the system interacts with actors without worrying about the details of how that functionality is implemented.



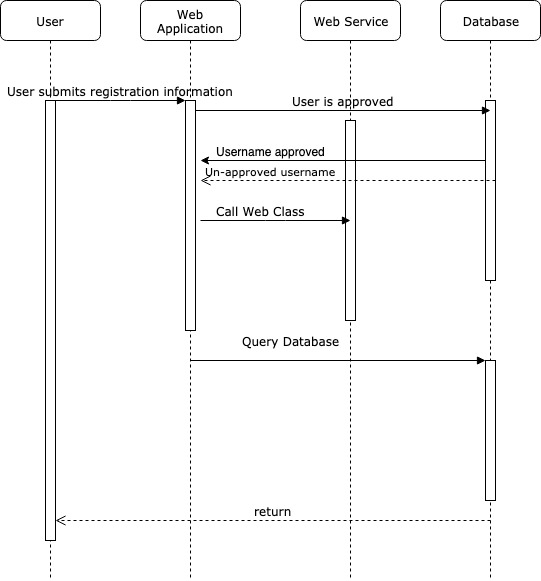
**Figure B.1.1: Use Case Diagram**

The use case is shown in figure A.4. It indicates the <<include>> and <<extend>> sub-use-case relationship. It also shows relation between player1 and player2 (player2 inherits all behaviors of player1). After registration, player has option to choose game mode: multiplayer or single player. In single player mode, player starts playing game right away. After lost, player has chance to buy more lives and continue playing. In multiplayer mode, player decides which opponent to challenge (from registered users list - in database). Chosen player (player 2) has choice to accept or refuse the challenge. If challenge is accepted, both players go to play game mode and start playing against each other. Game ends when one of the players loses. After lost, defeated player has a chance to buy more lives and play rematch.

**B.2: Sequence Diagrams for UC 1-8**

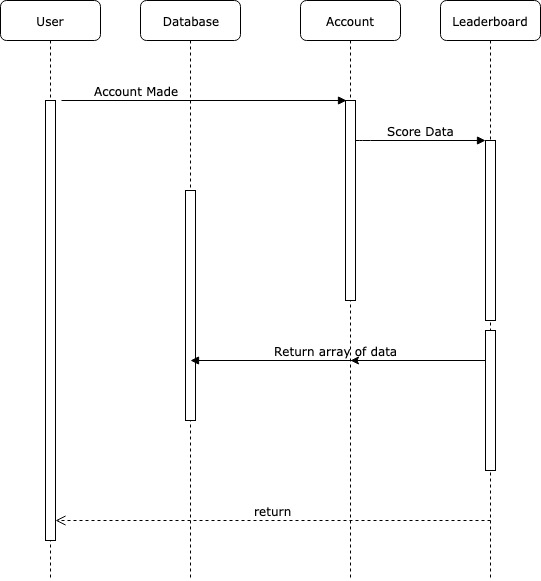
Sequence diagrams in UML are primarily used to model the interactions between the actors and the objects in a system and the interactions between the objects themselves. Sequence diagrams shows the sequence of interactions that take place during a particular use case or use case instance. They are a good way to visualize and validate various runtime scenarios. This can help to predict how a system will behave, as well as to discover responsibilities a class may need to have in the process of modeling a new system.

**Figure B.2.1: Sequence Diagram- Use Case 1A**

****

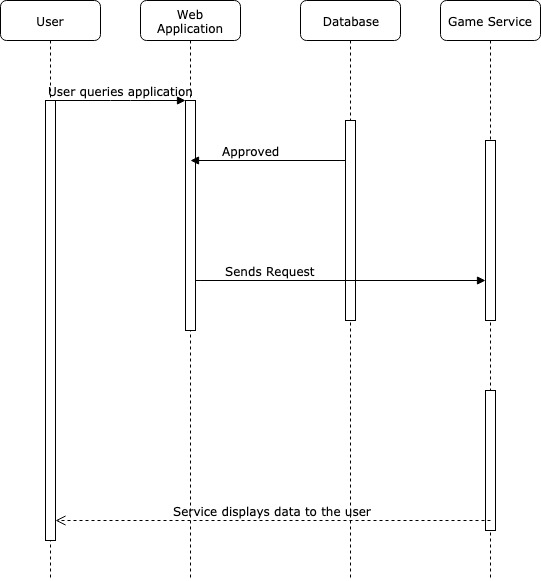
In Use Case 1, the User registers to play the video game. The User will enter the data and submit it on the Web Application. The Application queries the database to make sure the username is available. If the username is not available, the database calls the Web Application informing the user to create a new username. If the username is approved, the Application calls the Web Service class to process these inputs, and calls the Database to input the final username. The User now has an account and can play the game.

**Figure B.2.2: Sequence Diagram-Use Case 1B**

****

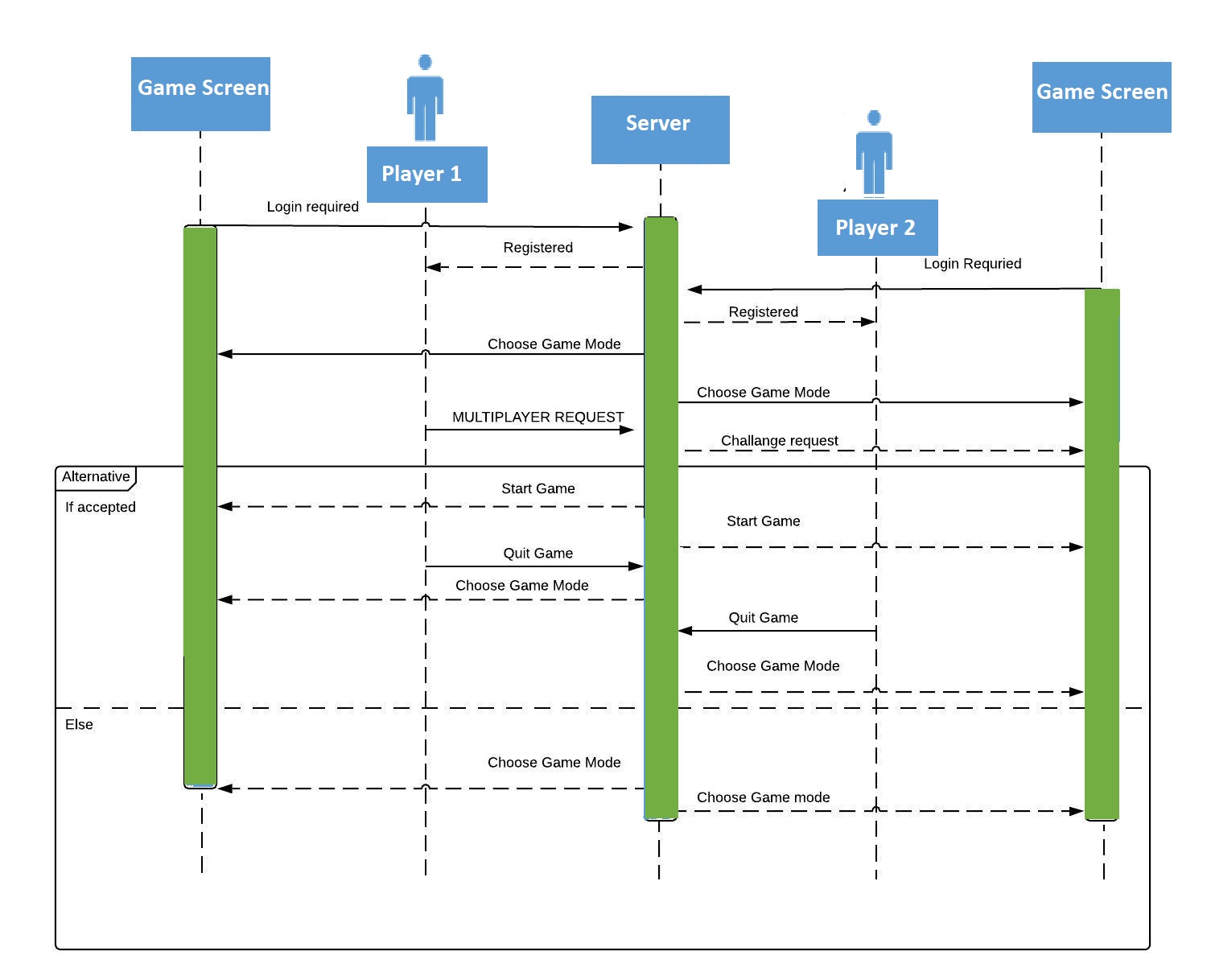
At this point the User’s account it made. The Account sends the User’s score data to the Leaderboard and the Leaderboard returns the array of data to the Account and the Database. Their score info will now be saved in the database.

**Figure B.2.3: Sequence Diagram-Use Case 2:**

****

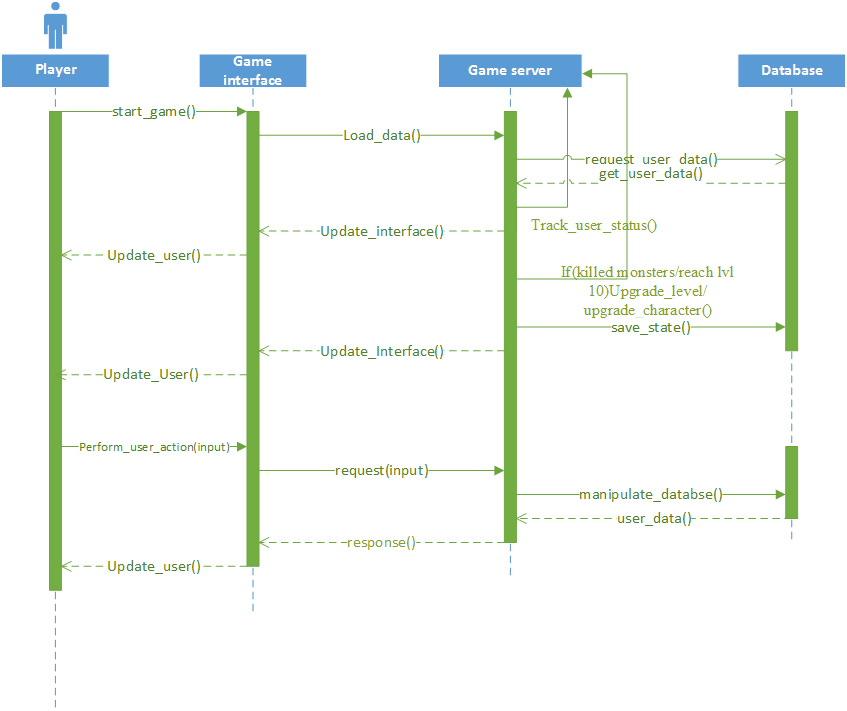
In Use Case 2, the User is allowed to challenge an opponent by entering their username into the video game’s database. The Database approves the two usernames. The web application then sends a request to the game service that this User wants to challenge another User. Lastly, the game service sends all the information back to the User, allowing the Users to compete.

**Figure B.2.4: Sequence Diagram-Use Case 3**

****

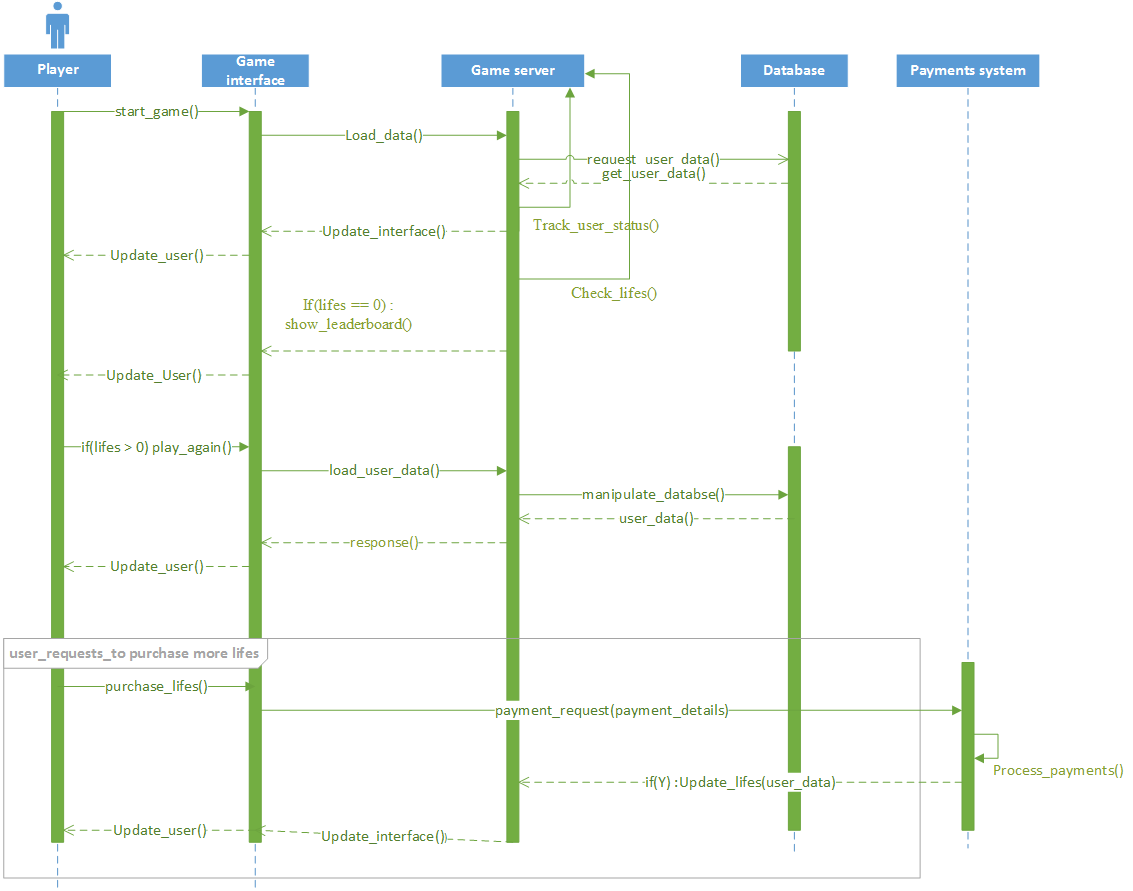
It is continuation of use case number 2. After login into the game, user has to register. After registration, user has option to choose game mode. When at least two players are available online, one of the players can request multiplayer mode. As a response, database sends request to the user that has been challenged. The second user has alternative to accept or reject the challenge. If the challenge is accepted, both players start game. During the game each player has the option to quit the game. If the multiplayer request is denied, both players go back into choose game mode.

**Figure B.2.5: Sequence Diagram-Use Case 4**

****

In the start, the player navigates to the game interface and requests for starting game. The player’s request is forwarded to the game server which loads the user data and updates the game’s graphical interface with the loaded user data and updates the user. Before updating the interface, an asynchronous call is made to game server itself for tracking the user state, which keeps track of user, life, levels e.t.c. and takes the player to character upgrade or to the next level if all monsters are killed. A player can perform action via interacting with graphical user interface, player inputs(p,q and space bar) are noted and a request is dispatched to the game server for responding to user inputs which manipulates the database and responds back updating the user via game interface.

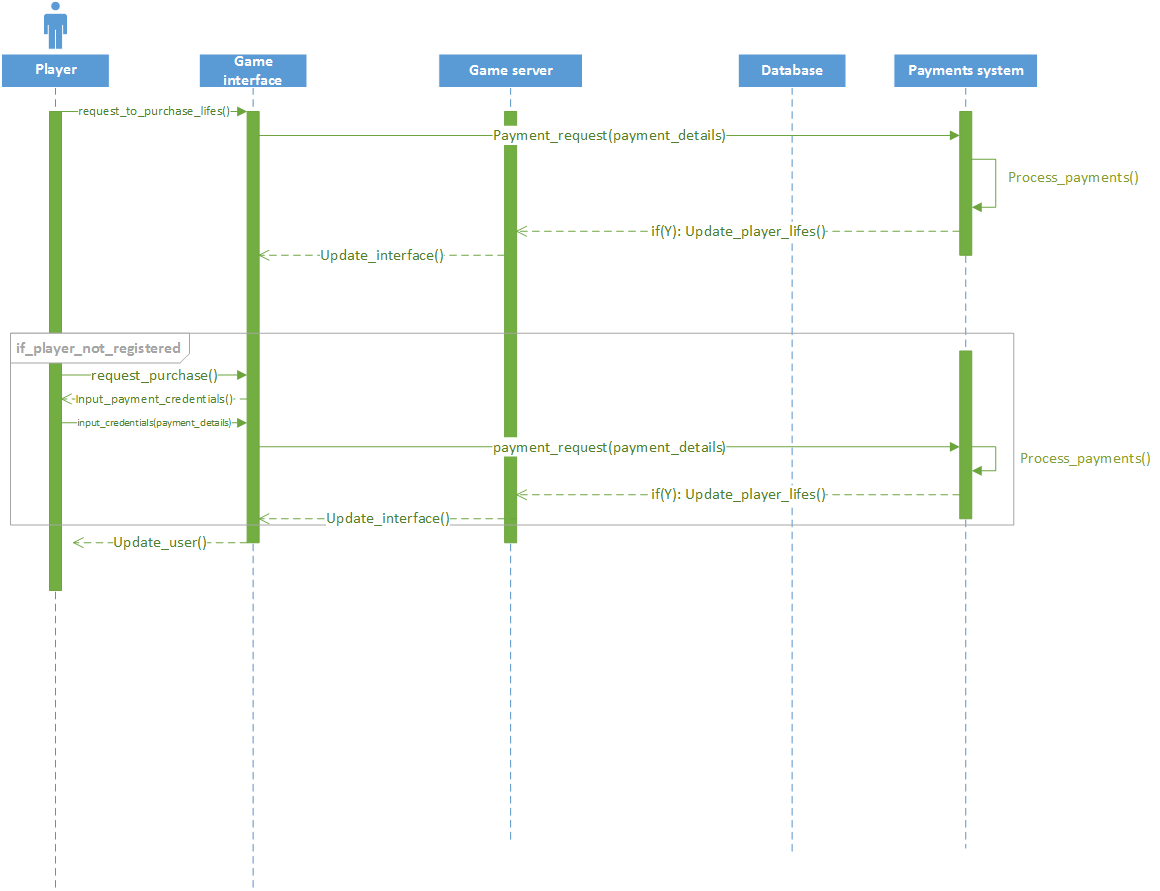
**Figure B.2.6: Sequence Diagram-Use Case 5**

****

In the start the player navigates to the game interface and requests for starting game, the player’s request is forwarded to the game server which loads the user data, and updates the game’s graphical interface with the loaded user data and updates the user. Before updating the interface, an asynchronous call is made to game server itself for tracking the user state, which keeps track of user and lives. Once the player’s life decreases to zero, player is taken to the leaderboard page.

If players life’s are greater than 0, a player can request to restarting the game or a player can request to purchase new lives via game interface, request is forwarded to payments system with the player’s payment credentials, which validates and process the payments, and if successful , requests the game server to update user lives. Lastly, the game interface is updated with changes and user is informed.

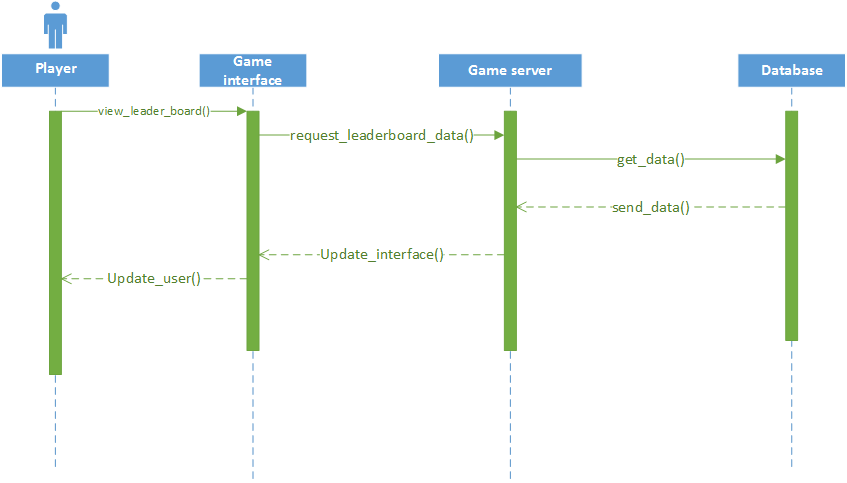
**Figure B.2.7: Sequence Diagram-Use Case 6**

****

Player can request to purchase new lives via game interface, request is forwarded to payments system with the player’s payment credentials, which validates and process the payments, and if successful, requests the game server to update user lives, and then game interface is updated with changes and user is informed.

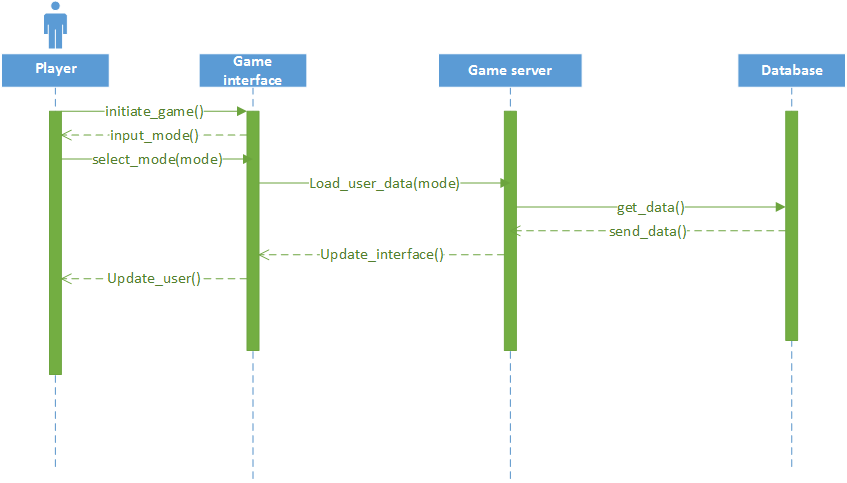
If a player is not registered, player is required to input payments credentials, which are then forwarded to payments system, and same process happens again.

**Figure B.2.8: Sequence Diagram-Use Case 7**

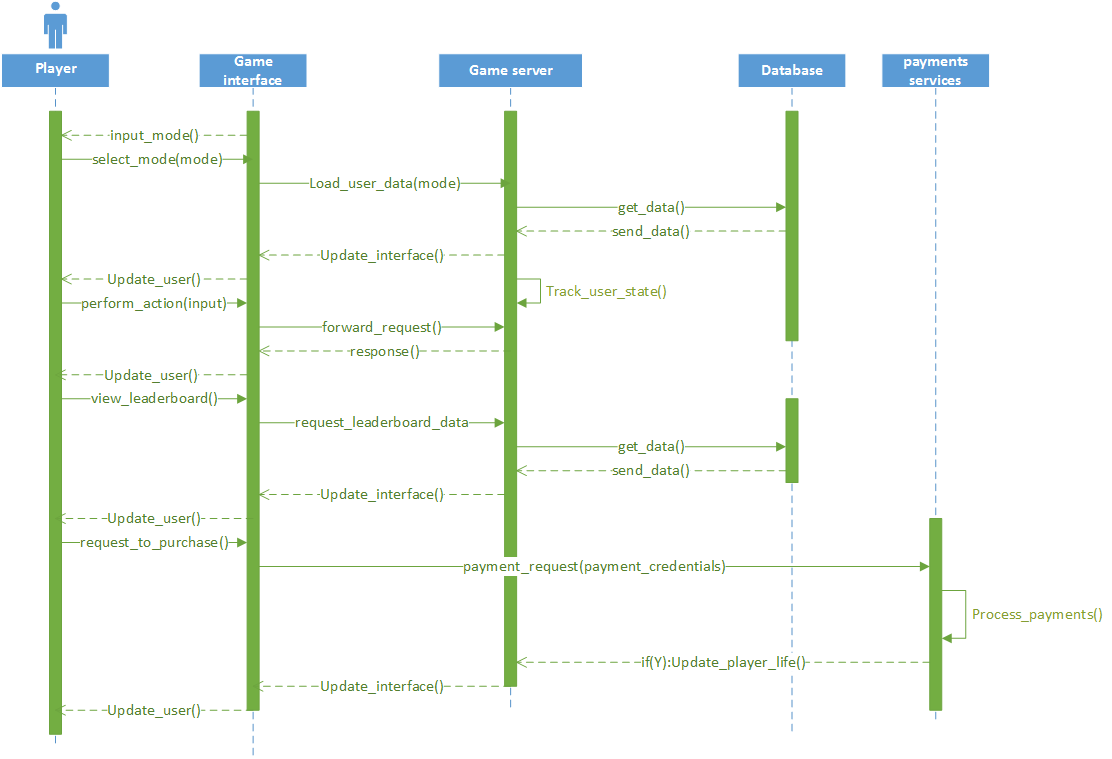
****

A player can request to view leaderboard via game interface, request is forwarded to game server which loads the leader board data from database and dispatches it to game interface, and then updating the user with this data.

**Figure B.2.9: Sequence Diagram-Use Case 8**

****

Player requests to initiate the game through game interface, before initiation player is required to select the game mode, after selection of game mode, request is sent to the game server with ‘mode’ parameter , which on the base of this parameter loads the player(s) data from database and then updating the game interface with this new data.

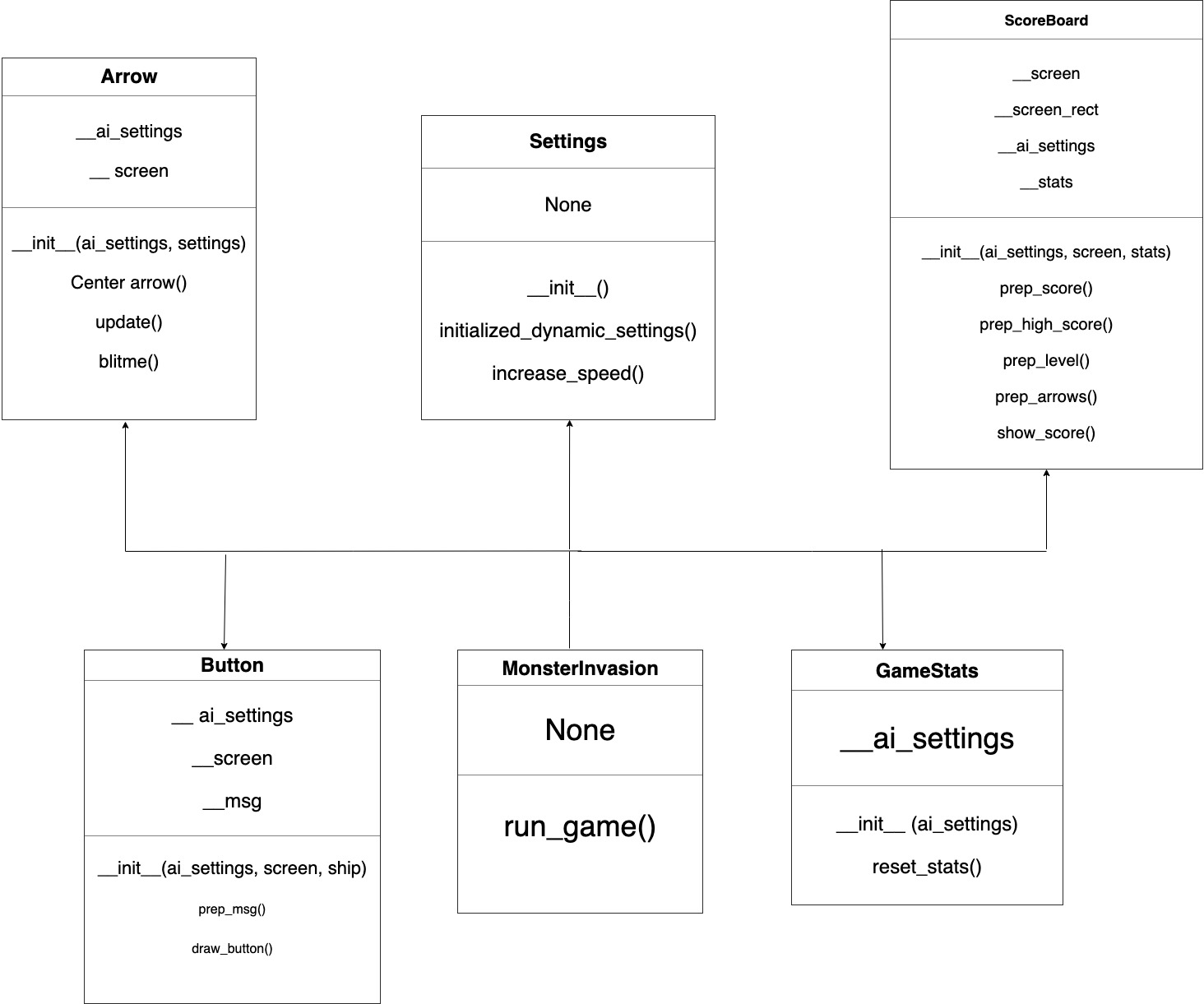
**Figure B.2.10: Sequence Diagram-Use Case 9 **

Describes all the processes took place from UC4 - UC8.

**Figure B.3.1: Class Diagram**

Class diagrams represent the static view of an application. They are not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

They describe the attributes and operations of a class and also the constraints imposed on the system. Class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.



**Figure B.3.1: Class Diagram**

The class MonsterInvasion is the child class and its parent classes include: Arrow, Button, Settings, GameStats, and ScoreBoard. MonsterInvasion inherits methods and attributes from all of those parent classes and uses them to execute.

**B.4: Architectural Diagrams**

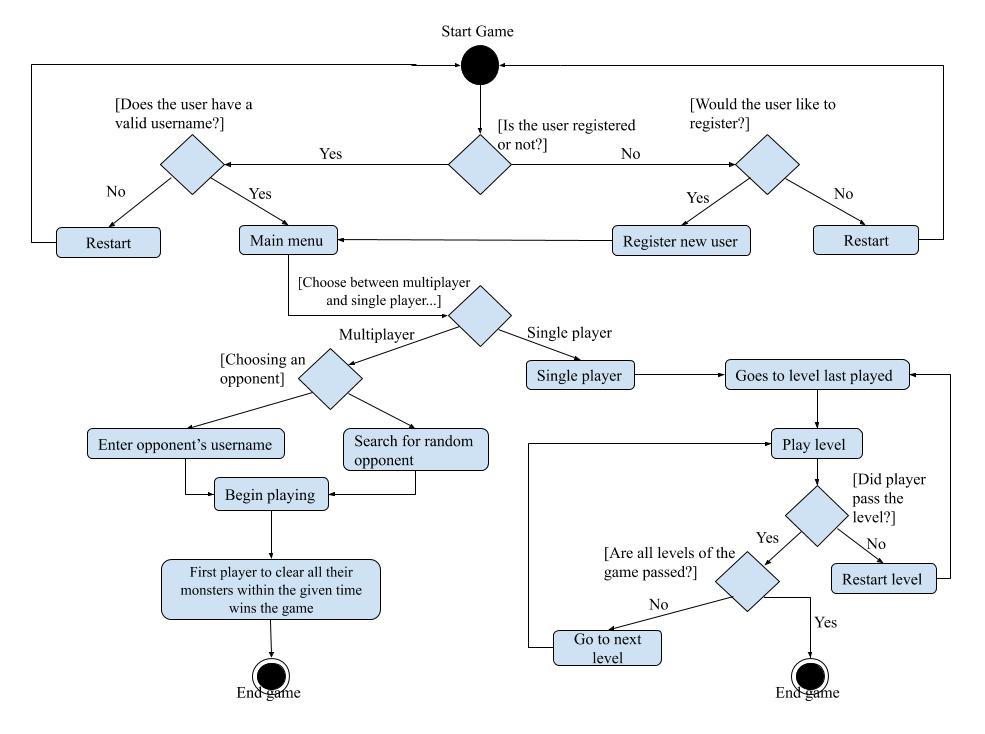
Architectural diagrams are concerned with how a system is organized, as well as the structure of that system. It is connecting design and the requirements engineering. The architectural diagrams we chose to create for our software are activity diagram, client-server diagram, and a layered diagram.

Activity diagrams visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. They also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. Either way, an activity diagram will have a beginning (an initial state) and an end (a final state). The diamonds represent a decision that is to be made and the rectangular boxes represent actions.

A client-server diagram is a distributed communication framework of network processes among service requestors, clients and service providers. A server manages most processes and stores all data. A client requests specified data or processes. The server then relays the process output to the client. Clients sometimes handle processing, but require server data resources for completion.

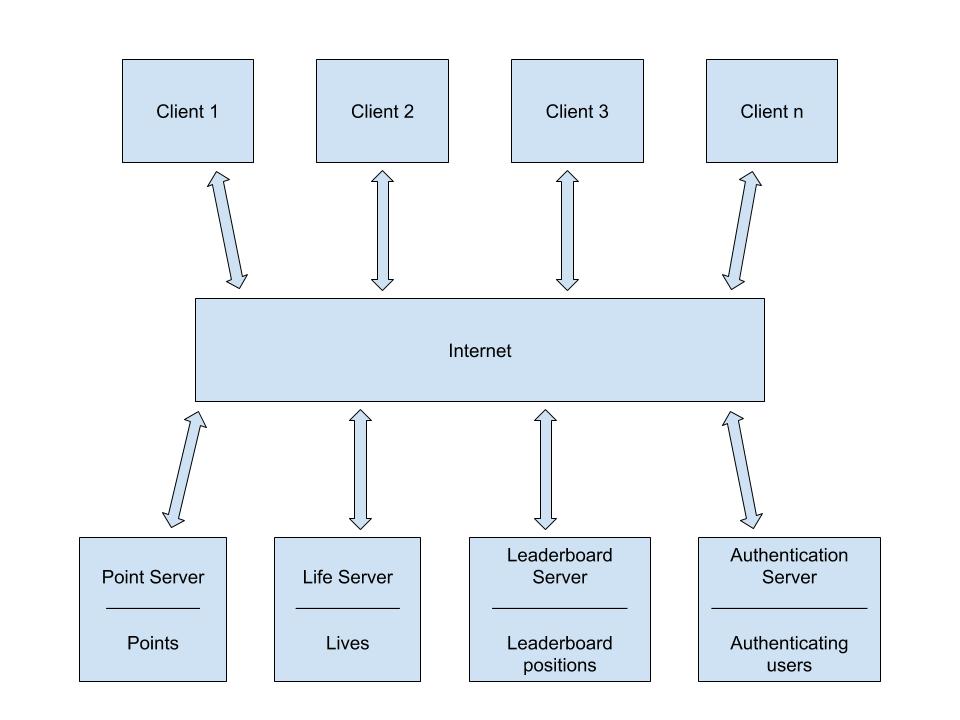
A layered diagram organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system.

**Figure B.4.1: Activity Diagram**

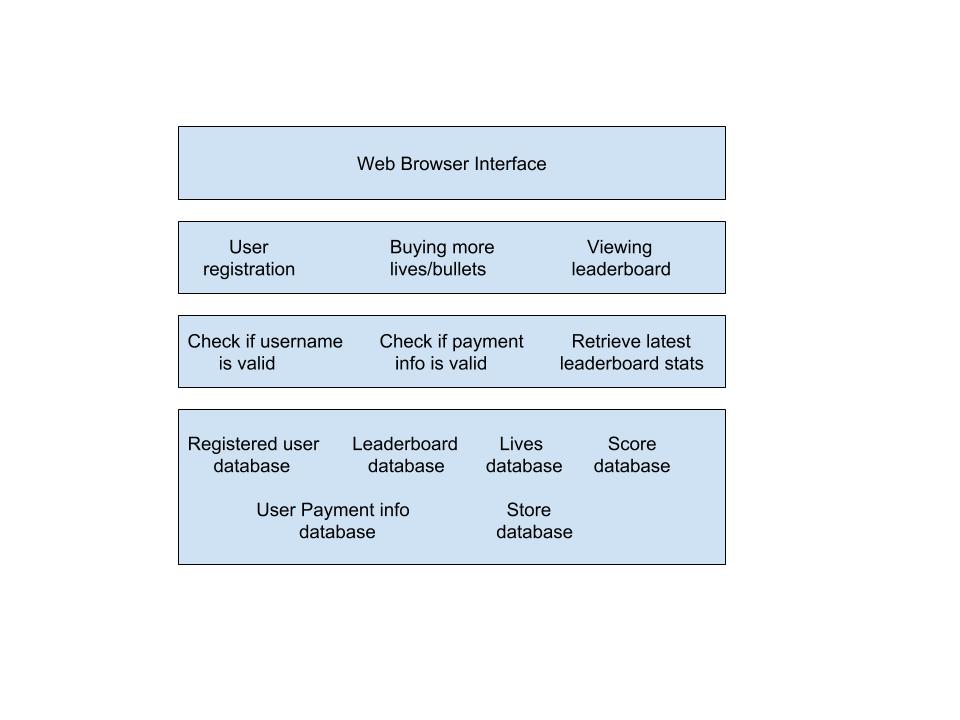
****

Our activity diagram visually depicts the flow of the game. Similar to a step-by-step tutorial of how our software runs.

**Figure B.4.2: Client-Server Diagram**

Client-server architecture is the depiction of how multiple clients can access the services of each server the system has to offer. In our diagram all of our clients are to gain access to these servers through the internet.

**Figure B.4.3: Layered Diagram**

****

Layered diagrams organize the system into different layers with related functionality that goes along with each layer. Each layer provides services to the layer before it, which makes the lowest layer crucial to the entire software.

**B.5: Test Cases**

Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use. When you test software, you execute a program using artificial data. The testing process has two goals. First is to demonstrate to the developer and the customer that the software meets the requirements and second to discover situation in which the behavior of the software is incorrect, undesirable, or does not conform to its specification. Testing can only show the presence of errors, not their absence.

Test cases are a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application.

**Table B.5.1: Test for UC1A:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case # | Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| 01 | Check response when user registers to play video game with new valid username(lookupUsername) | “R@chelle7” | System should provide whether the username is valid | System accepts the valid username with at least one capital letter, one special symbol, and one number | pass |
| 02 | Check response when user registers to play video game with invalid username(lookupUsername) | “rachelle7” | System should provide that this username is not valid | System detects error, displays message (“wrong input value) | fail |
| 03 | Check response when no input is provided(lookupUsername) | “ ” | System should detect invalid input and inform user | System detects error, displays message (“wrong input value) | fail |

**Table B.5.2: Test Case for UC1B:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case # | Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| 01 | Check response when user’s account is validated(lookupUsername) | “R@chelle7” | System shall send the user’s score data to the leaderboard | Leaderboard accepts the user’s data and sends it to the account database, detects errors, displays message(“wrong input value) | pass |
| 02 | Check response when user’s account is denied(lookupUsername) | “rachelle7” | System shall provide that this account is not valid and data can not be sent to leaderboard | Leaderboard does not accept any data, detects erros, displays message(“wrong input value) | fail |
| 03 | Check response when no input is provided (lookupUsername) | “ ” | System shall provide that this account is not valid and data can not be sent to leaderboard | Leaderboard does not accept any data, detects errors, displays message(“wrong input value) | fail |

**Table B.5.3: Test Case for UC2**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case # | Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| 01 | Check response when user registers to play video game with new valid username(lookupUsername) | “R@chelle7” | System should provide whether the username is valid | System accepts the valid username with at least one capital letter, one special symbol, and one number | pass |
| 02 | Check response when user registers to play video game with invalid username(lookupUsername) | “rachelle7” | System should provide that this username is not valid | System detects error, displays message (“wrong input value) | fail |
| 03 | Check response when no input is provided(lookupUsername) | “ ” | System should detect invalid input and inform user | System detects error, displays message (“wrong input value) | fail |

**Table B.5.4: Test Case for UC3:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case #** | **Description** | **Test Data** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| 01 | Test multiplayer mode – challenge request | “Player2” | System shall invite player2 into the multiplayer game with the challenger | Both interfaces (player1 and player2) receive notification about multiplayer challenge | pass |
| 02 | Test multiplayer mode – challenge acceptance | Challenger  Accepts  Game  invitation | System shall display multiplayer interface for both players | Server correctly updates interfaces for both players | pass |
| 03 | Test multiplayer mode – challenge refusal | Challenger  refuses  Game  invitation | System shall go back to single player mode | Server correctly updates interfaces for both players | pass |
| 04 | Test, in multiplayer mode – quit game | “Esc” | System shall display “Are you sure want to quit? message “ | System provides user with correct response | pass |

**Table B.5.5: Test Case for UC4:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case #** | **Description** | **Test Data** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| 01 | Test, playing game | Left arrow | Game character moves left across the screen (while left arrow is pressed) | Game character moves correctly across the screen | pass |
| 02 | Test, playing game | Right arrow | Game character moves left across the screen (while right arrow is pressed) | Game character moves correctly across the screen | pass |
| 03 | Test, playing game | space | Game character shoots 1 bullet | Game character shoots 1 bullet at the time when | pass |
| 04 | Test, playing game | Player shoots 1 monster | System shall update user score with +50 points. Killed monster should disappear from game interface | System correctly updates score and interface | pass |
| 05 | Test, playing game | Number of lives #1 | System shall provide new player with 4 lives. P | System provides new user with 1 live | fail |
| 05a | Test, playing game | Number of lives #2 | Player should lose 1 life when monsters reach the bottom of the screen | System correctly updates lives number | pass |
| 05b | Test, playing game | Number of lives #3 | every 15 minutes player should get 1 extra live | System correctly updates lives number | pass |
| 06 | Test, playing game | “p” | Interface – game paused | System correctly updates interface | pass |
| 07 | Test, playing game | “zwx\*\*\*\*\_” | Interface – should not react if any other key than ‘p’, ‘esc’, ‘space’ or arrows pressed | System does not response | pass |
| 08 | Test, playing game | hen player kills all monsters without losing all four lives | Interface – should provide user with the next level | System does not response | Fail |

**Table B.5.6: Test Case for UC5:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case #** | **Description** | **Test Data** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| 01 | Test payment request | Player request to order lives | System shall provide user with payment system (payment credentials) | Server correctly updates interface | pass |
| 02 | Test payment request credentials | Data provided correct credential information | System shall accept and process payment | System accepts and processes payment | pass |
| 02b | Test payment request credentials | Data provided, incorrect credential information | System shall not accept payment | System does not accept payment | pass |
| 03 | Payment accepted – lives update | n/a | System shall update number of lives accordingly to payment 1live = 1$, and inform user | Server correctly updates number of lives, user informed | pass |
| 04 | Test payment request credentials | Data provided correct credential information | System shall accept and process payment | System accepts and processes payment | pass |

\*\*\* Payment system is processed by PayPal. No additional test needed as PayPal provides protection for merchants.

**Table B.5.7: Test Case for UC7:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case #** | **Description** | **Test Data** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| 01 | Test leaderboard request (through interface) | Player request to see leaderboard interface via interface | System shall display leaderboard with 20 top scores ( data collected from database) | Server displays incorrect results (list of top 10 players displayed) | Fail |

**Table B.5.8: Test Case for UC8:**

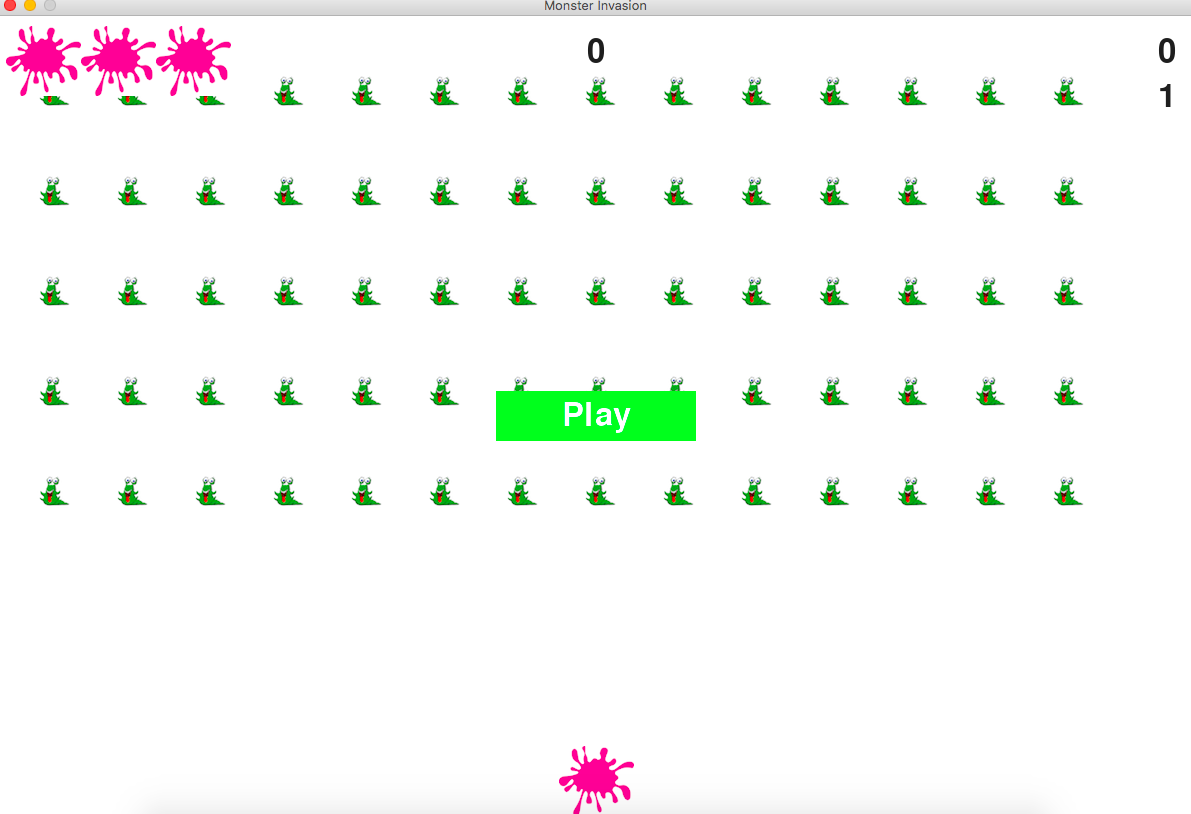
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case #** | **Description** | **Test Data** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| 01 | Test game initiation - single player (through interface) | Player selects multiplayer game mode | System shall display multiplayer interface | System does update game interface (not implemented) | Fail |
| 02 | Test game initiation - multiplayer (through interface) | Player selects single player game mode | System shall display single player interface | System correctly updates game interface with single player mode | Pass |

**Part 3 – Appendix C**

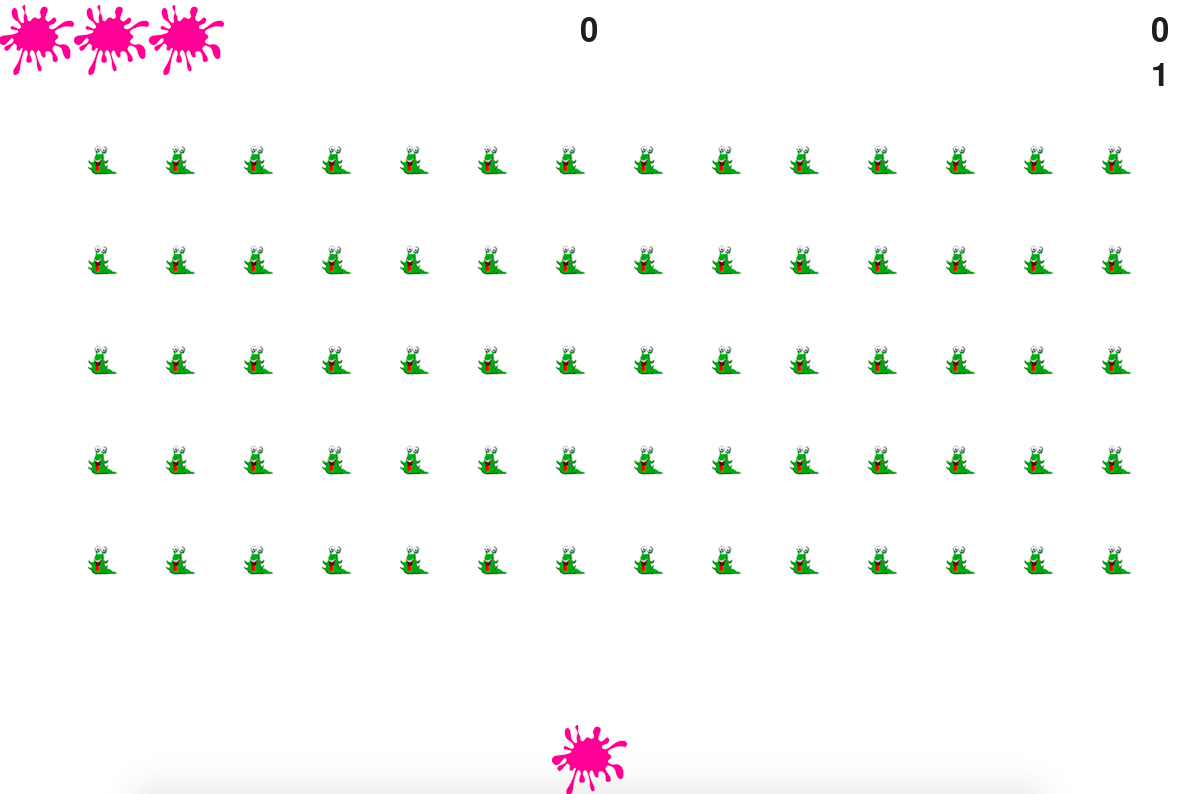
**C.1 Game Visuals**

**C.1.1: Game screenshots:**

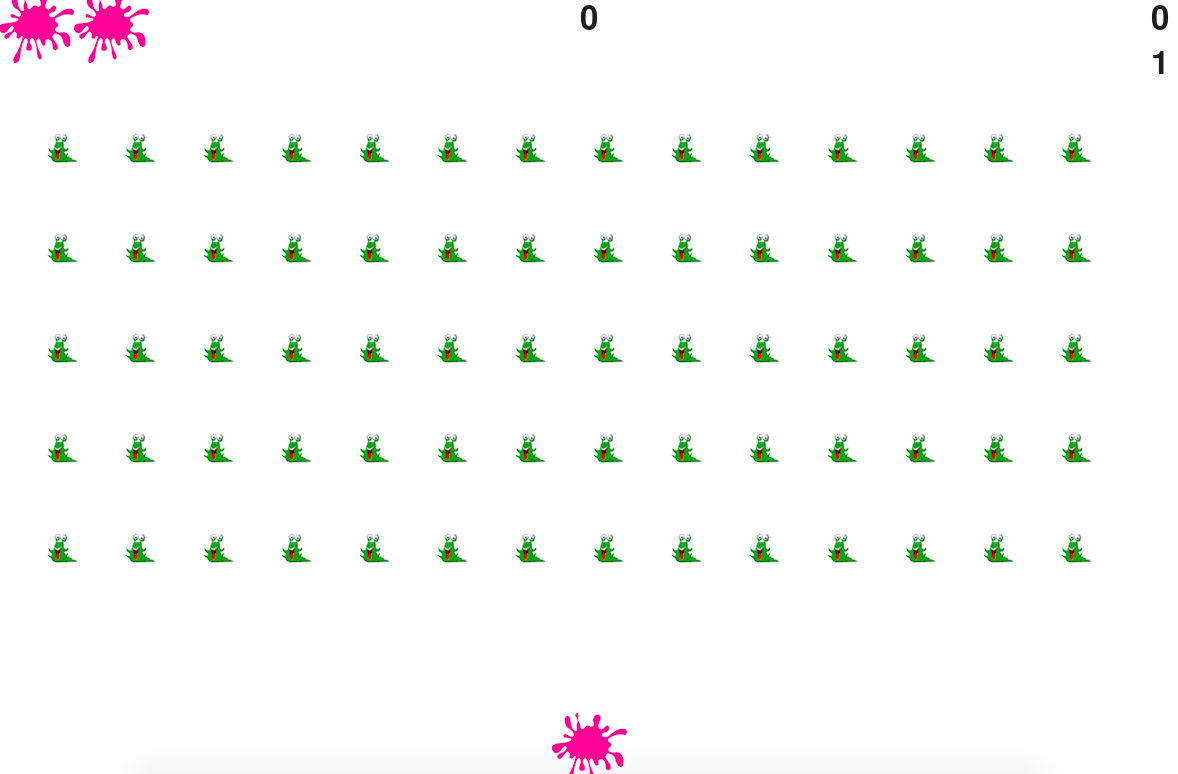
This photo is an example of the “play screen” when Monster Invasion is run in the terminal. To play the game, the user must press play.



This photo is an example of what the interface looks like after the user presses play. The user begins with 3 lives and a full army of monsters. The game begins at zero points.



This photo is an example of the interface when the user loses a level which results in the loss of a life.



**C.1.2 Screen Recording**

<https://youtu.be/dldlwZuxEpo>

**Part 5 – Appendix D**

**D.1 Source Code**

**D.1.1: Source Code: Arrow**

import pygame

from pygame.sprite import Sprite

class Arrow(Sprite):

def \_\_init\_\_(self, ai\_settings, screen):

#initialize the arrow and set its starting position

super(Arrow, self).\_\_init\_\_()

self.screen = screen

self.ai\_settings = ai\_settings

#movement flag

self.moving\_left = False

"""Initialize the arrow and set its starting position."""

#load the arrow image and get its rect

self.image = pygame.image.load('/Users/rachellerunyon/Desktop/slime2.bmp')

self.rect = self.image.get\_rect()

self.screen\_rect = screen.get\_rect()

#start each new arrow at bottom center of the screen

self.rect.centerx = self.screen\_rect.centerx

self.rect.bottom = self.screen\_rect.bottom

#store each decimal value for the arrows center.

self.center = float(self.rect.centerx)

#movement flag

self.moving\_right = False

self.moving\_left = False

def center\_arrow(self):

"""center the arrow on the screen"""

self.center = self.screen\_rect.centerx

def update(self):

"""Update the arrows position based on the movement flag."""

#update the arrow's center value, not the rect

if self.moving\_right and self.rect.right < self.screen\_rect.right:

self.center += self.ai\_settings.arrow\_speed\_factor

if self.moving\_left and self.rect.left > 0:

self.center -= self.ai\_settings.arrow\_speed\_factor

#update rect object from self.center

self.rect.centerx = self.center

def blitme(self):

"""draw the arrow at its correct location"""

self.screen.blit(self.image, self.rect)

**D.1.2: Source Code: Bullet**

import pygame

from pygame.sprite import Sprite

class Bullet(Sprite):

"""A class to manage bullets fired from the ship."""

def \_\_init\_\_(self, ai\_settings, screen, ship):

"""Create a bullet object at the arrow’s current position"""

super().\_\_init\_\_()

self.screen = screen

#create a cullet rect at (0,0) and then set correct position

self.rect = pygame.Rect(0, 0, ai\_settings.bullet\_width,ai\_settings.bullet\_height)

self.rect.centerx = ship.rect.centerx

self.rect.top = ship.rect.top

#store that bullets position as a decimal val

self.y = float(self.rect.y)

self.color = ai\_settings.bullet\_color

self.speed\_factor = ai\_settings.bullet\_speed\_factor

def update(self):

"""move the bullet up the screen"""

#update the decimal position of the bullet

self.y -= self.speed\_factor

#update the rect position

self.rect.y = self.y

def draw\_bullet(self):

"""draw the bullet to the screen"""

pygame.draw.rect(self.screen, self.color, self.rect)

**D.1.3: Source Code: Button**

import pygame.font

class Button():

def \_\_init\_\_(self, ai\_settings, screen, msg):

"""Initialize button attributes"""

self.screen = screen

self.screen\_rect = screen.get\_rect()

#set the dimensions and properties of the button

self.width,self.height = 200, 50

self.button\_color = (0, 255, 0)

self.text\_color = (255, 255, 255)

self.font = pygame.font.SysFont(None, 48)

#build the buttons rect object and center it

self.rect = pygame.Rect(0,0,self.width, self.height)

self.rect.center = self.screen\_rect.center

#The button message needs to be prepared only once

self.prep\_msg(msg)

def prep\_msg(self, msg):

"""Turn msg into a rendered image and center text on the button"""

self.msg\_image = self.font.render(msg, True, self.text\_color, self.button\_color)

self.msg\_image\_rect = self.msg\_image.get\_rect()

self.msg\_image\_rect.center = self.rect.center

def draw\_button(self):

#Draw blank button and then draw message.

self.screen.fill(self.button\_color, self.rect)

self.screen.blit(self.msg\_image, self.msg\_image\_rect)

**D.1.4: Source Code: game\_functions**

import pygame

import sys

from time import sleep

from bullet import Bullet

from Monster import Monster

def check\_keydown\_events(event, ai\_settings, screen,arrow,bullets):

"""Respond to key presses."""

if event.key == pygame.K\_RIGHT:

arrow.moving\_right = True

elif event.key == pygame.K\_LEFT:

arrow.moving\_left = True

elif event.key == pygame.K\_SPACE:

fire\_bullet(ai\_settings, screen, arrow, bullets)

elif event.key == pygame.K\_q:

sys.exit()

def check\_keyup\_events(event, arrow):

"""Respond to key releases."""

if event.key == pygame.K\_RIGHT:

arrow.moving\_right = False

elif event.key == pygame.K\_LEFT:

arrow.moving\_left = False

def check\_events(ai\_settings,screen, stats, sb, play\_button, arrow, monsters,

bullets):

"""Respond to key presses and mouse events."""

for event in pygame.event.get():

if event.type == pygame.QUIT:

sys.exit()

elif event.type == pygame.KEYDOWN:

check\_keydown\_events(event, ai\_settings, screen, arrow,bullets)

elif event.type == pygame.KEYUP:

check\_keyup\_events(event,arrow)

elif event.type == pygame.MOUSEBUTTONDOWN:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

check\_play\_button(ai\_settings, screen, stats, sb, play\_button,

arrow, monsters, bullets, mouse\_x, mouse\_y)

def check\_play\_button(ai\_settings, screen, stats, sb, play\_button, arrow,

monsters, bullets, mouse\_x, mouse\_y):

"""Start a new game when the player clicks Play."""

button\_clicked = play\_button.rect.collidepoint(mouse\_x, mouse\_y)

if button\_clicked and not stats.game\_active:

# Reset the game settings.

ai\_settings.initialize\_dynamic\_settings()

# Hide the mouse cursor.

pygame.mouse.set\_visible(False)

# Reset the game statistics.

stats.reset\_stats()

stats.game\_active = True

# Reset the scoreboard images.

sb.prep\_score()

sb.prep\_high\_score()

sb.prep\_level()

sb.prep\_arrows()

# Empty the list of aliens and bullets.

monsters.empty()

bullets.empty()

# Create a new fleet and center the ship.

create\_fleet(ai\_settings, screen, arrow, monsters)

arrow.center\_arrow()

def fire\_bullet(ai\_settings, screen, arrow,bullets):

"""Fire a bullet if limit not reached yet."""

# Create a new bullet and add it to the bullets group.

if len(bullets) < ai\_settings.bullets\_allowed:

pygame.mixer.init()

pygame.mixer.music.load('/Users/rachellerunyon/Desktop/blast.wav')

pygame.mixer.music.play()

new\_bullet = Bullet(ai\_settings, screen, arrow)

bullets.add(new\_bullet)

def update\_screen(ai\_settings, screen, stats, sb, arrow, monsters, bullets,

play\_button):

"""Update images on the screen and flip to the new screen."""

# Redraw the screen during each pass through the loop.

screen.fill(ai\_settings.bg\_color)

arrow.blitme()

monsters.draw(screen)

# Redraw all bullets behind ship and aliens.

for bullet in bullets.sprites():

bullet.draw\_bullet()

# Draw the score information.

sb.show\_score()

# Draw the play button if the game is inactive.

if not stats.game\_active:

play\_button.draw\_button()

# Make the most recently drawn screen visible.

pygame.display.flip()

def update\_bullets(ai\_settings, screen, stats, sb, arrow, monsters, bullets):

"""Update position of bullets and get rid of old bullets."""

# Update bullet positions.

bullets.update()

# Get rid of bullets that have disappeared.

for bullet in bullets.copy():

if bullet.rect.bottom <= 0:

bullets.remove(bullet)

# print(len(bullets))

check\_bullet\_monster\_collisions(ai\_settings, screen, stats, sb, arrow, monsters, bullets)

def check\_high\_score(stats, sb):

"""Check to see if there's a new high score."""

if stats.score > stats.high\_score:

stats.high\_score = stats.score

sb.prep\_high\_score()

def check\_bullet\_monster\_collisions(ai\_settings, screen, stats, sb, arrow, monsters, bullets):

"""Respond to bullet-alien collisions."""

# Remove any bullets and aliens that have collided.

collisions = pygame.sprite.groupcollide(bullets, monsters, True, True)

if collisions:

for monsters in collisions.values():

stats.score += ai\_settings.monster\_points \* len(monsters)

sb.prep\_score()

check\_high\_score(stats, sb)

if len(monsters) == 0:

bullets.empty()

ai\_settings.increase\_speed()

#inc level

stats.level += 1

sb.prep\_level()

create\_fleet(ai\_settings, screen, arrow, monsters)

#start\_new\_level(ai\_settings, monsters, bullets, sb, screen, arrow, stats)

def check\_fleet\_edges(ai\_settings, monsters):

"""Respond appropriately if any aliens have reached an edge."""

for monster in monsters.sprites():

if monster.check\_edges():

change\_fleet\_direction(ai\_settings, monsters)

break

def change\_fleet\_direction(ai\_settings, monsters):

"""Drop the entire fleet and change the fleet's direction."""

for monster in monsters.sprites():

monster.rect.y += ai\_settings.fleet\_drop\_speed

ai\_settings.fleet\_direction \*= -1

def arrow\_hit(ai\_settings, screen, stats, sb, arrow, monsters, bullets):

"""Respond to arrow being hit by monster."""

if stats.arrows\_left > 0:

# Decrement ships\_left.

stats.arrows\_left -= 1

**D.1.5: Source Code: game\_stats**

class GameStats():

"""Track stats for monster invasion"""

def \_\_init\_\_(self, ai\_settings):

"""initialize stats"""

self.ai\_settings = ai\_settings

self.reset\_stats()

#start monster invasion in an inactive state.

self.game\_active = False

#high score should never be reset

self.high\_score = 0 #attributes

def reset\_stats(self):

"""Initialize stats that can change during the game"""

self.arrows\_left = self.ai\_settings.arrow\_limit

self.score = 0

self.level = 1

**D.1.6: Source Code: monster**

import pygame

from pygame.sprite import Sprite

class Monster(Sprite):

"""A class to represent a single monster in the fleet """

def \_\_init\_\_(self, ai\_settings, screen):

"""Initialize the monster and set its starting position."""

super(Monster, self). \_\_init\_\_()

self.screen = screen

self.ai\_settings = ai\_settings

#load the monster image and set its rect attribute

self.image = pygame.image.load('/Users/rachellerunyon/Desktop/monster3.bmp')

self.rect = self.image.get\_rect()

#start each new monster near the top left of the screen

self.rect.x = self.rect.width

self.rect.y = self.rect.height

#store the monsters exact position

self.x = float(self.rect.x)

def check\_edges(self):

"""Return True is monster us at edge of screen"""

screen\_rect = self.screen.get\_rect()

if self.rect.right >= screen\_rect.right:

return True

elif self.rect.left <= 0:

return True

def update(self):

"""Move the monster right or left"""

self.x += (self.ai\_settings.monster\_speed\_factor \* self.ai\_settings.fleet\_direction)

self.rect.x = self.x

def blitme(self):

"""Draw the monster at its current location"""

self.screen.blit(self.image, self.rect)

**D.1.7: Source Code: monster\_invasion**

import pygame

from pygame.sprite import Group

from settings import Settings

from game\_stats import GameStats

from scoreboard import Scoreboard

from button import Button

from arrow import Arrow

import game\_functions as gf

def run\_game():

pygame.init()

ai\_settings = Settings()

screen = pygame.display.set\_mode(

(ai\_settings.screen\_width, ai\_settings.screen\_height))

pygame.display.set\_caption("Monster Invasion")

#Make the play button

play\_button = Button(ai\_settings, screen, "Play")

#create an instance to store game stats and create a scoreboard

stats = GameStats(ai\_settings)

sb = Scoreboard(ai\_settings, screen, stats)

#make an arrow, a group of bullets,and a group of monsters

arrow = Arrow(ai\_settings, screen)

#make a group to store bullets in

bullets = Group()

monsters = Group()

gf.create\_fleet(ai\_settings, screen, arrow, monsters)

print('here')

arrow.blitme()

#start the main loop for the game

while True:

gf.check\_events(ai\_settings, screen, stats,sb, play\_button, arrow, monsters, bullets)

if stats.game\_active:

arrow.update()

gf.update\_bullets(ai\_settings, screen,stats,sb, arrow, monsters, bullets)

gf.update\_monsters(ai\_settings, screen, stats, sb, arrow, monsters, bullets)

gf.update\_screen(ai\_settings, screen, stats, sb, arrow, monsters, bullets, play\_button)

**D.1.8: Source Code: scoreboard**

run\_game()

import pygame.font

from pygame.sprite import Group

from arrow import Arrow

class Scoreboard():

"""A class to report scoring info"""

def \_\_init\_\_(self, ai\_settings, screen, stats):

"""Initialize scorekeeping attributes"""

self.screen = screen

self.screen\_rect = screen.get\_rect()

self.ai\_settings = ai\_settings

self.stats = stats

#Font settings for scoring info

self.text\_color = (30, 30, 30)

self.font = pygame.sysfont.SysFont(None, 48)

#Prepare the initial score image

self.prep\_score()

self.prep\_high\_score()

self.prep\_level()

self.prep\_arrows()

def prep\_score(self):

"""Turn the score into a rendered image"""

rounded\_score = int(round(self.stats.score, -1))

score\_str = "{:,}".format(rounded\_score)

# score\_str = str(self.stats.score)

self.score\_image = self.font.render(score\_str, True, self.text\_color, self.ai\_settings.bg\_color)

#Display the score at top right of screen

self.score\_rect = self.score\_image.get\_rect()

self.score\_rect.right = self.screen\_rect.right - 20

self.score\_rect.top = 20

def prep\_high\_score(self):

"""Initialize the high score into a rendered image"""

high\_score = int(round(self.stats.high\_score,-1))

high\_score\_str = "{:,}".format(high\_score)

self.high\_score\_image = self.font.render(high\_score\_str,True, self.text\_color, self.ai\_settings.bg\_color)

#center the high score at top of the screen

self.high\_score\_rect = self.high\_score\_image.get\_rect()

self.high\_score\_rect.centerx = self.screen\_rect.centerx

self.high\_score\_rect.top = self.score\_rect.top

def prep\_level(self):

"""Turn the level into a rendered image"""

self.level\_image = self.font.render(str(self.stats.level), True, self.text\_color, self.ai\_settings.bg\_color)

#position the level below the score

self.level\_rect = self.level\_image.get\_rect()

self.level\_rect.right = self.score\_rect.right

self.level\_rect.top = self.score\_rect.bottom +10

def prep\_arrows(self):

"""Show how many ships are left"""

self.arrows = Group()

for arrow\_number in range(self.stats.arrows\_left):

arrow = Arrow(self.ai\_settings, self.screen)

arrow.rect.x = 10 + arrow\_number \* arrow.rect.width

arrow.rect.y = 10

self.arrows.add(arrow)

def show\_score(self):

"""Draw score and arrows to the screen"""

self.screen.blit(self.score\_image, self.score\_rect)

self.screen.blit(self.high\_score\_image, self.high\_score\_rect)

self.screen.blit(self.level\_image, self.level\_rect)

#draw arrows

self.arrows.draw(self.screen)

**D.1.9: Source Code: settings**

class Settings():

"""A class to store all settings for Monster Invasion."""

def \_\_init\_\_(self):

"""Initialize the game's static settings."""

#screen settings

self.screen\_width = 1200

self.screen\_height = 800

self.bg\_color = (255, 255, 255)

#arrow settings

self.arrow\_limit = 3

#bullet settings

self.bullet\_width = 8

self.bullet\_height = 15

self.bullet\_color = 220, 40, 154

self.bullets\_allowed = 10

#monster settings

#self.monster\_speed\_factor = 1

self.fleet\_drop\_speed = 10

#How quickly the game speeds up

self.speedup\_scale = 1.0

#how quickly the monster point vals inc

self.score\_scale = 1.5

self.initialize\_dynamic\_settings()

def initialize\_dynamic\_settings(self):

"""Initialize settings that change throughout the game"""

self.arrow\_speed\_factor = 4

self.bullet\_speed\_factor = 3

self.monster\_speed\_factor = 5

#fleet\_direction of 1 represents right; -1 represents left

self.fleet\_direction = 1

#scoring

self.monster\_points = 50

#fleet\_irection of 1 represents right; -1 represents left.

self.fleet\_direction =1

def increase\_speed(self):

"""Increase speed settings and monster point values"""

self.arrow\_speed\_factor \*= self.speedup\_scale

self.bullet\_speed\_factor \*= self.speedup\_scale

self.monster\_speed\_factor \*= self.speedup\_scale

self.monster\_points = int(self.monster\_points \* self.score\_scale)

#print(self.monster\_points)

**Part 6 – Appendix E**

**E.1 References**

Sommerville Software Engineering 9th edition

file:///Users/sedaalioglu/Downloads/Project%20Sample%20(7).pdf