Software Engineering CS440

Project Report

Battleship Extreme

GROUP-9

Basil Issac

Shubham Jain

Kanishka Garg

Bradley Robinson

University of Illinois at Chicago

Table of Contents

List of Figures

I. Project Description

1 Project Overview

This game is a 2-D board game based on the conventional Battleship game, but with new features and strategies. The user starts the program through his or her web browser. A menu will welcome the player along with asking for his or her information, like name, age and country for playing online multiplayer and keeping high score. After entering their information, another menu will appear prompting them to select “Human vs. Computer” or “Human vs. Human”. If in one player mode, “Human” will select grid size varying from 10x10, 15x15, or 20x20. If in two player mode, “Human” that initiated the game will select grid size, and will send invitation to another player to join the game. Two grids will be created for each player. The lower grid will contain the player’s ships and other resources along with red “X”s for opponents hits , and the top grid will track the player’s moves against his or her opponent with red “X”s for player’s HITs and blue “O”s for player’s MISS. The screen will also contain a notification table which contains opponent’s activities and resources. The main goal of this game is to destroy all of the opponent's ships.

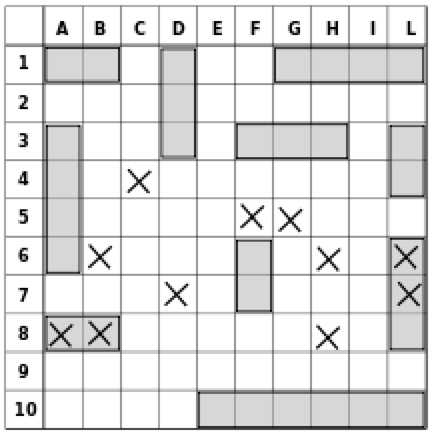


Fig.1(a): layout of a classic battleship game

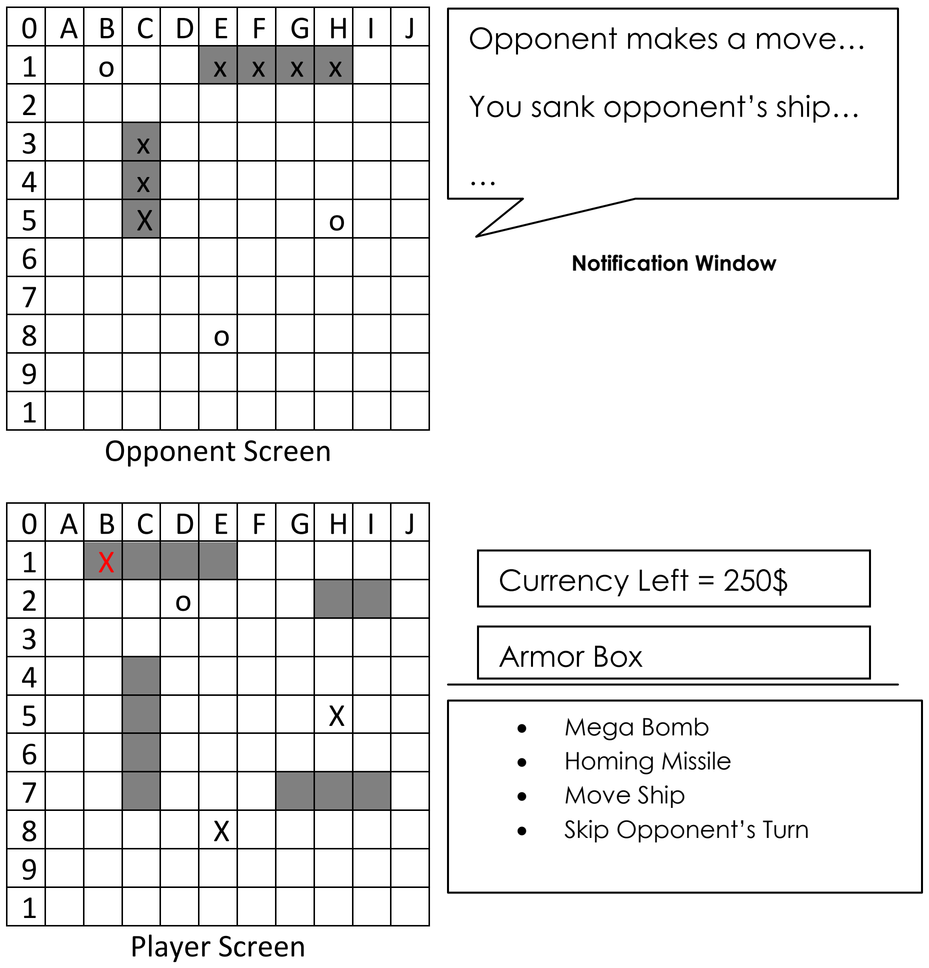


Fig. 1(b): Our version of Battleship Game

The rules to play Battleship Extreme are primarily based on the standard Battleship rules with a twist. Before the game starts, the user is instructed to place the ships in various locations on the board. The game is then played by the user and another human, or the CPU guessing where the opposition's ships are on the board. Players will take turns making an attack which is done by selecting a square; the program will then look in the opponent's square and determine if the inputted guess is a "HIT" or a "MISS". If the player chooses correctly (hits), the corresponding square on top grid is marked with a red "X" and they are then able to attack again. If the player chooses incorrectly, the corresponding square on top grid is marked with a blue "O" and their turn ends. The game is won by the user who strategically guessed the location of all of the opposing player's ships and destroys them.

This version of battleship includes a twist based on currency and purchasing resources. In the beginning of the game, each player is given a certain amount of money. With this money, each player is able to purchase unique items, such as radars, homing missiles, special bombs, etc. on their turn. A player is able to generate revenue through two different ways. The first way a player can generate revenue is through an “Ore Refinery”. Ore refineries are place able pieces (that take up 2x2 squares) available for purchase which generate income at the end of each move (HIT or MISS). In the beginning of the game, each player is issued a single Ore Refinery to start them off. But be careful, because they are place able pieces, Ore Refineries are also vulnerable to attack; leaving the player with no steady stream of revenue should the opponent sink all of their refineries. Also, having an Ore Refinery is not completely the same as having a ship on the field of play! If a player sinks all of the opponent’s ships, yet leaves the Ore Refineries untouched, the player still wins.

The second method to generate revenue is to HIT the opponent’s ships, and sinking a whole ship will generate more revenue. Successfully sinking an opponent’s ship nets you a certain amount of money based on the ship sunk. More valuable (bigger) ships sunk earn bigger payoffs! The only consequence for getting a MISS will result in player’s turn being over.

The revenue generated from the previous methods can be used to make purchases on various attack weapons, like various bombs, homing missiles, and radars. It can also be spent on defensive methods, like relocating the player’s ships or getting an extra turn. Each attack weapon or defensive method will cost a certain price according to its usefulness (ex. radar will be less than a bomb, since radar does no damage). Each weapon will have a different use, which will be described later.

2 Domain

The game falls under the category of ‘Action Strategy’ board games. Action strategy titles combine the planning and management of strategy games with fast-paced action or shooter elements. This style applies when the player can leave the strategic overview to take direct control in action sequences, such as attacking the opponent’s ships while still in command of saving his own ships in the field. Typically, strategy elements such as building structures, creating units, and managing resources are also part of the gameplay.

The game is more complex when including grid size, currency, and more attack and defensive elements. There has to be two types of AI, one which checks if player is playing by the rules, and another AI to make decisions for the Computer in one player mode. However, for one player mode, the AI has to make decisions based on the computer’s currency, past moves, and resources of both players. At certain points of the game the computer will have to make “strategic” guesses based on the combined information of hits, misses, grid size, and probability of ship locations. It would be beneficial to consult an expert to calculate probability, so the AI can make intelligent moves.

3 Hardware and Software

The game will be developed using the java language allowing it to be compatible with a wide variety of platforms. Any libraries or other files needed by the game will be included with the game. The hardware that is needed in order to play the game is any system that can run Java, a monitor to display the gameboard/instructions, a mouse to enter the commands, and a keyboard to enter user information. Any hardware resources like a special graphics card/controller is not needed as the game will be developed with simple graphics initially. There is no need of any special CPU requirements but, as long as the machine is able to run java, our game is able to run. A speaker or earphones is needed if a player wants to play sound in the background of the game. Any modern OS like Windows 7/8/XP or Linux or MacOS are our targeted platforms and we want our game to be compatible with modern browsers like IE 9.0, Firefox, Google Chrome and Safari. In future, the game can be developed for mobile-based OS like Android, iOS, or Windows.

The IDE which we plan to use for the development of source code is Eclipse, as it is widely used throughout the world for developing java-based applications. The other major and pivotal tool which we plan to use is git for the source-code management. We plan to follow an agile methodology for the development of this project and plan to use open-source tools like IceScrum for monitoring the activities of scrum teams during the sprints. We plan to use Google Drive as a collaboration tool in order to exchange the development artifacts. We will primarily be testing this game against functional requirements on windows platform.

4 Client

The game vendors of social networking sites like Facebook, MySpace etc which have hosted apps for multiplayer / single games would serve as our primary target clients for this game. Zynga is one of the major game development companies around the world, which develops such games for social networking clients. The company focuses on delivering free, small, quick to learn, games of all types of genres to the world via web browser. Board game is a special category which Zynga is trying to follow up, since there are lots of board game lovers around the world. Moreover, board games involve more strategic thinking, than first person shooter games. Hence, intriguing users of all ages would be interested to play a game having warfare dynamics implemented on a board game. Battleship Extreme is a board game involving critical thinking with strategic moves, war elements with various armory attacks, multiplayer mode of playing the game worldwide, and the option to play against a computer with pre-defined AI rules and algorithms etc. Battleship Extreme will give a new dimension to the board games with its unique style of play, and we feel that game vendors like Zynga will support this game to intrigue the board game lovers in the near future. This game will keep the spirit of board games alive in the fast moving digital age.

5 End User

The primary target for this game is the users of age 12+ who have interest in the games involving strategy. The game deals with the aspects of money management and decision-making, so a minimum level of calculation skills like addition of numbers is required. The user should be familiar and know how to use a computer with keyboard and mouse. The game would be appended as an application with social-networking sites like Facebook, MySpace etc. Thus, end users are required to have an internet access along with an account on the corresponding social networking website. The users should be able to understand English language as this game will have rules about how to play written in English. This game is expected to elevate the problem-solving skills, management skills and decision-making skills of the-end-users.

6 Team

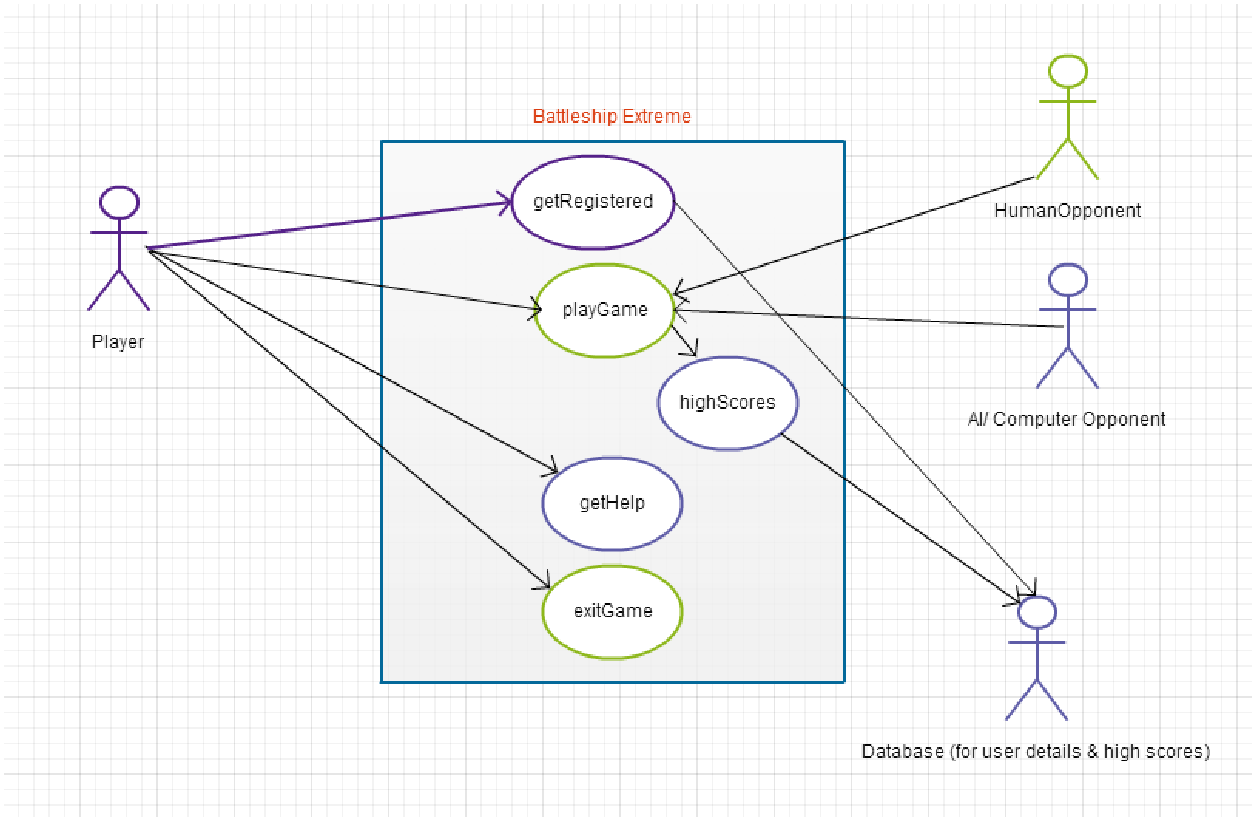
The team could be made up of four people. The team shouldn’t have any hierarchical order, as that is one of the basic principles of Agile methodology. The team should plan on developing the work in sprints and dividing the work into two releases comprising of two 4-week sprints. The team should be divided into two teams called “scrum” teams and plan to follow the “Pair programming” technique of Agile Software Development in which two programmers work together at one workstation. In Pair programming, one, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The two programmers should switch roles frequently. The team can rely on collaborative tools like GroupMe, WhatsApp, Git, IceScrum, Google Hangouts or Google docs for meeting and coordinating purposes.

7 Use Case

Use Case Name	PlayGame
Participating Actors	Initiated by Player(s).
Flow of Events	1. The Player initiates the game by selecting it on the website. 2. The game displays a starting menu asking the Player for his/her name, age, country. 3. The Player enters the required info and selects play option. 4. The Player then decides if they want to play another human or the CPU. 5. The game shows the option to select board size. 6. The Player selects what size board they'd like to play on. 7. The game displays a blank board, asking user to set its ships on board. 8. The Player sets their pieces and selects start option 9. The game starts
Entry Condition	The Player chooses the game from the list of games displayed on the webpage.
Exit Condition	The Players board is loaded on the screen with initial currency set to 1000 units.

8 Use Case Diagrams

The use case diagram above shows the user playing the game against either a human or CPU opponent with the game being hosted on the web server. This is an extremely simple diagram showing how the user interacts with the Battleship Extreme program. As you can see, the user only has the option to play the game or not, and the user has the option to choose between another human playing the game or a CPU. The game is, of course, played on the web server and the web server keeps track of the prompted information from the user in the opening screen; it also keeps track of whether or not the user won the game.



II. Requirements

9 Functional Requirements

9.1 The game shall support proper human input. The user will be able to input information through keyboard use.

Rationale: So that user can create an account and easily navigate the game.

Fit Criterion: The information shall be entered through the keyboard and main screen shall be displayed. The mouse will primarily be used to control the main functions of the game.

9.2 The game shall support an intelligent AI for the Human vs. Computer mode. The AI shall follow certain rules and make moves based on the opponent's move, keeping track of his own past moves and the probability of making a HIT.

Rationale: So that the user can play even if another human Player in order to play is not available and intelligent AI in order for the game to be challenging enough for the user to be able to enjoy it.

Fit Criterion: The computer shall destroy/make a move on its turn against the Player.

9.3 The game shall support virtual currency for purchasing the war inventories for both the players.

Rationale: Virtual Currency is the new functionality in this game for strategy and Purchasing from inventory with virtual currency is one of the virtual currency's use.

Fit Criterion: Virtual currency counter shall be displayed on the play screen and shall be Updated when Player buys any weapon.

9.4 There shall be a pre-defined cost associated with every weapon to be bought during the game. The cost of each weapon will be determined by the amount of the damage caused by that weapon.

Rationale: Different prices for different type of weapons are also a strategy element which forces the users to make strategy in order to minimize losses and maximize profits. Higher investment yields better returns so cost is directly proportional to the effectiveness of weapon bought.

Fit Criterion: Attack Mechanism Select screen shall be displayed and there shall be a cost associated with every weapon. The cost of weapons shall be in the following order: Giga Bombs > Mega Bombs > Homing Missile > Radar.

9.5 The game shall keep track of the currency held by both players throughout the gameplay. The ore refineries will generate money at every move, until the game is over, unless it is destroyed by the opponent player. The game will be able to update the total currency for each player on every move made. Apart from purchasing weapons, the virtual money could also be used to move the player's own ship or skip the opponent's turn, as a defense mechanism.

Rationale: Another use of Virtual Currency. Selecting what to do with the money is part of the strategic features and hence another use. Currency is an important field of the game which controls a lot of other features and hence, its tracking is necessary.

Fit Criterion: Virtual Currency field for each Player shall be updated after every move by the any of the Player. The virtual currency counter shall be increased by 20 units, which is generated by the ore refineries.

9.6 The game shall support proper human input. The user will be able to input information through keyboard use. The mouse will primarily be used to control the main functions of the game.

9.7 The game shall support an intelligent AI for the Human vs Computer mode. The AI shall follow certain rules and make moves based on the opponent's move, keeping track of his own past moves and the probability of making a HIT.

9.8 The game shall support virtual currency for purchasing the war inventories for both the players. There shall be a pre-defined cost associated with every weapon to be bought during the game. The cost of each weapon will be determined by the amount of the damage caused by that weapon. The game shall keep track of the currency held by both players throughout the gameplay. The ore refineries will generate money at every move, until the game is over, unless it is destroyed by the opponent player. The game will be able to update the total currency for each player on every move made. Apart from purchasing weapons, the virtual money could also be used to move the player's own ship or skip the opponent's turn, as a defense mechanism.

9.9 The game shall have an area to notify the player of the opponent's move. The notification area will help the player strategize their future move. The game shall maintain the logs for all the notifications to be displayed in the notification area.

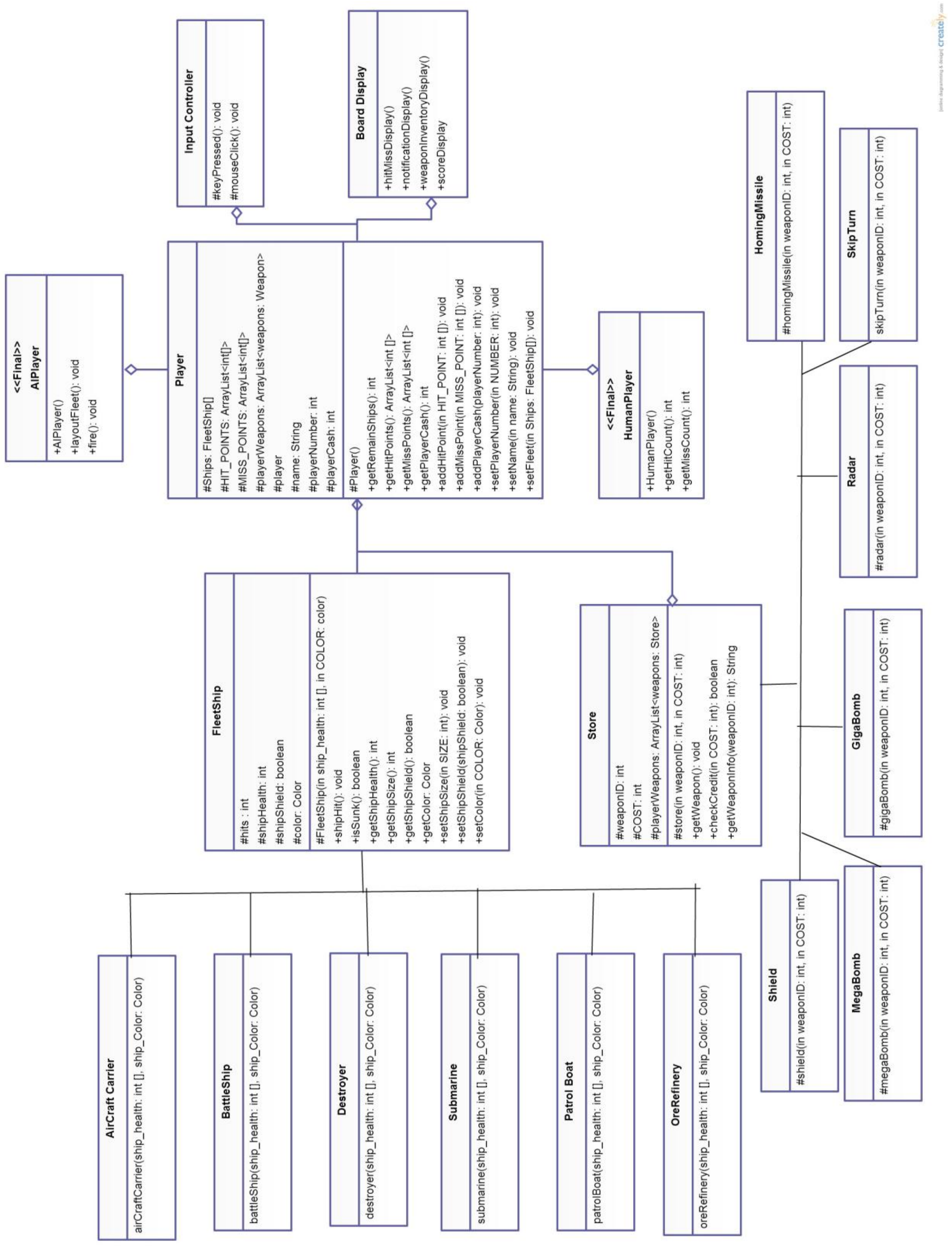
9.10 The game shall support various attack weapons such as Giga Bombs, Mega Bombs, Homing Missiles and Radars, each with a different attack mechanism. With the Giga bombs, the user will be able to select one grid space, and if it is a HIT, the whole ship will sink. The Mega-Bombs will explode within a 3x3 grid space and if any ship within range will be hit. The player will also be notified if it is a HIT or a MISS. Homing missiles will be used by selecting a 3x3 grid space but only hitting one section of a ship (at random), if it is in range. With the homing missile, the player will only be notified if it is a HIT or a full MISS. The radar will also be a 3x3 grid space that will notify the player of the ships in that area.

9.11 The game shall also support some defensive strategies to forestall the opponent's attack. There are three types of defensive methods that the game will support. First, when the opponent uses the radar and finds the player's ship, then the player can purchase "moves" to move their undamaged ship to a location that was not attacked (excluding radar) by their opponent. Second simple defense method we have decided to add is skipping the opponent's turn so the player can get another attack in if they MISS. Third, the player can use the "shield" to remain undetected by the opponent's homing missile attack.

9.12 The game shall maintain a database to manage each player's portfolio in the multiplayer support functionality. The portfolio will contain two tables to manage player's information. One table will store player's personal information like name, age, location. The other table will maintain high-scores for all the players.

9.13 The game shall support multiplayer functionality, so that it could be integrated on social networking websites to be played globally.

10 Data Requirements


Input Controller
#keyPressed(): void
#mouseClick(): void
Board Display
+hitMissDisplay()
+notificationDisplay()
+weaponInventoryDisplay()
+scoreDisplay
<<Final>>
AIPlayer
+AIPlayer()
+layoutFleet(): void
+fire(): void
Player
#Ships: FleetShip[]
#HIT_POINTS: ArrayList<int[]>
#MISS_POINTS: ArrayList<int[]>
#playerWeapons: ArrayList<weapons: Weapon>
#player
#name: String
#playerNumber: int
#playerCash: int
#Player()
+getRemainShips(): int
+getHitPoints(): ArrayList<int []>
+getMissPoints(): ArrayList<int []>
+getPlayerCash(): int
+addHitPoint(in HIT_POINT: int []): void
+addMissPoint(in MISS_POINT: int []): void
+addPlayerCash(playerNumber: int): void
+setPlayerNumber(in NUMBER: int): void
+setName(in name: String): void
+setFleet(in Ships: FleetShip[]): void
<<Final>>
HumanPlayer
+HumanPlayer()
+getHitCount(): int
+getMissCount(): int
HomingMissile
#homingMissile(in weaponID: int, in COST: int)
SkipTurn
skipTurn(in weaponID: int, in COST: int)
Radar
#radar(in weaponID: int, in COST: int)
GigaBomb
#gigaBomb(in weaponID: int, in COST: int)
MegaBomb
#megaBomb(in weaponID: int, in COST: int)
Shield
#shield(in weaponID: int, in COST: int)
FleetShip
#hits : int
#shipHealth: int
#shipShield: boolean
#color: Color
#FleetShip(in ship_health: int [], in COLOR: color)
+shipHit(): void
+isSunk(): boolean
+getShipHealth(): int
+getShipSize(): int
+getShipShield(): boolean
+getColor: Color
+setShipSize(in SIZE: int): void
+setShipShield(shipShield: boolean): void
+setColor(in COLOR: Color): void
Store
#weaponID: int
#COST: int
#playerWeapons: ArrayList<weapons: Store>
#store(in weaponID: int, in COST: int)
+getWeapon(): void
+checkCredit(in COST: int): boolean
+getWeaponInfo(weaponID: int): String
AirCraft Carrier
airCraftCarrier(ship_health: int [], ship_Color: Color)
BattleShip
battleShip(ship_health: int [], ship_Color: Color)
Destroyer
destroyer(ship_health: int [], ship_Color: Color)
Submarine
submarine(ship_health: int [], ship_Color: Color)
Patrol Boat
patrolBoat(ship_health: int [], ship_Color: Color)
OreRefinery
oreRefinery(ship_health: int [], ship_Color: Color)

11 Performance Requirements

- Speed and Latency
    - Game should be loaded within 10 seconds.
    - Screens in the game should be loaded within 3 seconds.
    - Command responses should take at most 150 milliseconds.
    - Opponents move should be made within 10 seconds.
    - Notification table should be updated within 200 milliseconds.
    - Currency and weapons purchases should be updated within 250 milliseconds
- Precision/Accuracy Requirements
    - Command parsing and execution, currency calculation and the opponent's execution should be accurate.
    - Currency displayed should be in terms of dollars
    - Currency should be incremented in multiples of 5 and must be positive integers.
- Capacity Requirements
    - The game should, at least, be able to handle up to 1000 players playing simultaneously.

12 Dependability Requirements

- Reliability and Availability Requirements
    - No data should be lost in the event of failure.
    - During maintenance, older versions of the game should be made available to the public until the current version is 100% operational.
    - The game should not fail more than 3 times a day.
    - The game should be available to play at all times.
    - Maintenance and upgrades should not last more than 24 hours.
    - If the game fails, it should take, at most, 20 seconds to reload.
- Robustness or Fault-tolerance Requirements
    - The game should allow the player to go to the control page if they've entered an invalid move.
    - The game should go into local mode for 5 minutes if the network goes down or internet disconnects.
- Safety-Critical Requirements
    - Playing the game should not result in any kind of damage to the player.
    - The game should not display rapidly changing images or flashing lights.

13 Maintainability and Supportability Requirements

- Maintenance Requirements
  - The game should have an ‘about us’ in the main menu
  - The game should be maintained by developers who are not the creators.
  - Patches for bugs should be released within 15 days after detection.
  - There should be an option to change the background music.

- Supportability Requirement
  - There should be help and support for the user base through email.
  - Users should get a response to any inquiry within 24 hours.
  - The users should also be able to receive support through telecommunications.

- Adaptability Requirement
  - The game should be able to run on any java enabled web browser.
  - The game should support up to the latest OS and be at least compatible with Windows XP (as well as Mac OS and Unix-like OS).

- Scalability or Extensibility Requirement
  - The game should be able to run with at least 1000 players in its initial release.
  - After 2 years, the game should be able to run with at least 5000 players.

- Longevity Requirement
  - As long as there are social networking websites, the game should be able to run.
  - There should not be a gap of more than 6 months between subsequent releases of the game.

14 Security Requirements

- Access Requirements
  - Access to functionality and data should be granted only to the developers and test managers.
  - Developers and mangers should not have access to sensitive data entered by the player.
  - Player's personal information should be only be accessible to the business development manager.
  - Any individual who is not a developer or manager should get the consent of management to gain access to functionality and game data.

- Integrity Requirements
  - The data should be accessible only by the database administrator.
  - At no point, should, any of the data be lost.
  - There should be a periodic check every 24 hours in order to ensure the integrity of data.

- Privacy Requirements
  - Privacy Policy should be located in the 'about us' tab.
  - The game should send an email to the player's email ID telling them of the information policy of the company.
  - No player should be allowed command or control over another player's actions.
  - The game should protect private information in accordance with the relevant privacy laws and the company's information policy.

- Immunity Requirements
  - The game should be scanned every 12 hours for malicious software.
  - The system should notify the development team if malicious software is detected.

15 Usability and Humanity Requirements

- Ease of Use Requirements
    - The game should be appropriate for users aged 12+.
    - The user should be comfortable with the controls within 6 minutes of gameplay.
    - The user should not be able to change the controls.
    - The user is expected to know how to read English.
    - The user should have the option to stop/mute the background music.

- Learning Requirements
    - The learning curve to Battleship Extreme should be very low.
    - The user should have an option to play a tutorial level.
    - The user is expected to be familiar with basic addition and subtraction for tracking currency.
    - The user is expected to possess basic mouse and keyboard skills and an account on a social network.

- Accessibility Requirements
    - The game should provide support for the partially blind.
    - The game's features must be enhanced to support people with disabilities dependent on its popularity.
    - The game must follow a set of game accessibility guidelines similar to the Web Content Accessibility Guidelines in order to cater people with disabilities like hearing & visual impairment, motor impairment, cognitive impairment etc.

16 Look and Feel Requirements

- Appearance Requirements
  - All objects should look like their real life counter parts.
  - The game must appear state of the art.
  - The music should be rich with melody and support a war-like theme.
  - The primary target for this game are the users of age 12+ with the main focus for teenagers. Hence, the game GUI should be attractive to teenagers.

- Style Requirements
  - The player must have the feeling that he/she is actually playing in a war.
  - The player must have continuous interaction with the game throughout the game-play.
  - Since, it is a remake of a board game; the user must also feel that he/she is playing the game on a board.
  - Stylistically, the game should appear to be an aggressive and more active version of the old board game of battleship.

17 Operational and Environmental Requirements

- Expected Physical Requirements
    - A computer that has internet access, a social network account a web browser with a java plugin, a keyboard and a mouse are all the necessary equipment to run Battleship Extreme.
    - The game will not be hosted on websites for which the user must pay to gain access and will be completely free to play.

- Release Requirements
    - A new version of the game will be released no later than 3 months (other than the bug fix releases).
    - Bug Fix will be developed within 15 days after the bug has been identified.
    - Each subsequent major release will focus on adding an extra functionality, improving visual features, or audio and video capabilities of the game.
    - The developers will increase the frequency of releases after 2 years of the initial release since the count of users shall increase as the game is expected to become popular with time. Each release should be done after proper testing including regression testing so that the old features will also work along with the new ones.

18 Cultural and Political Requirements

- Battleship Extreme should not be offensive to any religious or ethnic group.
- The game should not use any terminology/linguistics that is proprietary to any particular cultural area anywhere.
- The game should not use any terms or icons that might possibly offend anyone on the planet.
- The product must use American English for all the spellings.

19 Legal Requirements

The images and music used in the game must be purchased in order to avoid any copyright infringement issues.

III. System Models

20 Scenarios

Use Case Name	selectAttack
Participating Actors	Initiated by John:Player Mike:OpponentPlayer
Flow Of Events	1. John decides he wants to play a game of Battleship Extreme. So he loads the game up, looks at his options from the main menu, and selects play game also selecting between playing against another player or the AI. 2. BE loads John's board along with either Mike's board (to see the attacks already made) or the AI's board, depending on if John decides to play against the player or the computer. 3. John then decides he wants to attack the opposition. Since this is Battleship Extreme, John has a large variety of weapons to lay waste to his opponent, so he selects a weapon using the ***selectWeapon*** function. 4. BE will then display the weapon type John selected, printing out a nifty image of the weapon, it's strength and characteristics. 5. John, having selected his weapon, now needs to attack his enemy. He will then select an area, using ***selectAttArea***. 6. BE will then look at the weapon John is using and destroy the segment of the board John selected depending on the characteristics and strength of the weapon John has decided to use 7. John will get notified of whether he hit or miss the opposition in the notification area. If there is a human enemy, then Mike will be notified about the same using ***updateBoard.***
Entry Condition	It is John's turn, John selects "Attack" and the board is set-up
Exit Condition	John and Mike received the notifications and the game boards have been updated using ***updateBoard.***
Quality Requirements	a) The notification to John and Mike should come within 500 milliseconds. b) The notification should be consistent.

Fig 20(a): Use Case Scenario for selectAttack Use Case

Use Case Name	buyWeapon
Participating Actors	Initiated by John:Player
Flow of Events	1. John, looks at his userinterface and sees that he has accumulated a lot of money. There's really only one way to spend money in this game so John decides to actually use his accumulated currency and selects the option to buy a weapon 2. BE responds to John by showing a list of weapons with their corresponding price, attributes and strengths. 3. John looks at the list of weaponry and picks the weapon he feels will give him the highest chance of laying waste to his opponent. 4. BE will then look at the amount of money in John's account and the cost of the weapon. If John has less money than the cost of the weapon he has decided to buy, then BE will deny the purchase and offer the Player to select another weapon. Otherwise the corresponding weapon is purchased 5. BE will then check if John has enough space in John's inventory for the weapon. If BE sees that he has enough space for the weapon, it will be added to John's inventory. Otherwise the weapon will be returned to the shop and John's currency will be returned to him. 6. John's balance and Inventory is updated using ***updateBoard***. 7. BE responds by giving an option to ***selectAttack***.
Entry Condition	It is the John's turn and John selects "Buy"
Exit Condition	The weapon should be added into the inventory of the Player and cash balance is updated.
Quality Requirements	a.) Cash and balances should be correctly updated. b.) Weapons should not overlap in inventory

Fig 20(b): Use Case Scenario for buyWeapon Use Case

Use Case Name	inviteFriendandPlay
Participating Actors	Initiated by Mike:Player John:Opponents
Flow Of Events	1. Mike is sitting at home talking to a friend on the internet and they decide they want to play a game of Battlefield Extreme. Mike then loads the game up and selects the Player vs. Player mode. 2. BE responds by giving an option of selecting board size to Mike. 3. Mike selects the board size of 20x20 and John does the same, both having determined this as the size of the board they'd like to play with. 4. BE gives the list of Opponents with the same board size as Mike. Since Mike and John agreed to play with a 20x20 board size, they both will appear in the list of gamers playing with this board size. 5. Mike selects his friend John from the list given by the BE using ***selectOpponentPlayer***. 6. BE sends an invitation to join & play to John. 7. John receives the invitation to join and play the game. John can accept or decline the invitation request. Since John wants to play the game, he obviously accepts the invitation request. 8. BE will then read John's answer to the invitation request, in this instance John has accepted the invitation so BE loads the game. If John had declined the invitation, BE would have sent a message to Mike saying John declined and will allow Mike to reselect an opponent. OPTIONAL: 9. If the John declines the invitation, then select the opponent again using ***selectOpponentPlayer***.
Entry Condition	1. Mike is logged into the social networking website account on which the game is hosted. 2. Mike has greater than zero friends on the social networking website account. 3. Mike has selected Player vs. Player as the mode of play.
Exit Condition	1. The Player vs. Player game mode is loaded. OR 2. Mike plays with an AI in the case he/she has no friends to play with.
Quality Requirements	1. Mike is acknowledged within 1 sec. about the accept/decline of the invitation. 2. The game gets loaded within 2 seconds. 3. There should be a search function after the players decide the board size to make things easier for friends to find each other.

Fig 20(c): Use Case Scenario for inviteFriendandPlay Use Case

Use Case Name	selectPlayWithAI
Participating Actors	Initiated by John:Player
Flow Of Events	1. John wants to play a game of battleship extreme. None of John's friends available to play feel like playing this wonderful game having spent the last ten thousand plus nights straight playing it, so he selects the Player vs. AI(Computer) after loading up the game. 2. BE responds by giving an option of selecting board size to John. 3. John selects the board size, make things interesting by making a 100x100 board. 4. BE loads the game and the board is setup. 5. John plays the game against the AI.
Entry Condition	1. John is logged into the social networking website account on which the game is hosted. 2. John has selected Player vs. AI (Computer) as the mode of play
Exit Condition	The Player vs. AI game mode is loaded
Quality Requirements	1. The game gets loaded within 2 seconds. 2. AI completes its turn within 2 seconds.

Fig 20(d): Use Case Scenario for selectPlayWihAI Use Case

Use Case Name	winnerPvP
Participating Actors	Initiated by John:Player Mike:OpponentPlayer
Flow Of Events	1. John and Mike have played a few rounds of Battlefield Extreme. BE will then detect at the start of every round past 17 if there is a winner using ***detectWinner*** and if there is one prompts a winner screen to winner and loser screen to loser. It then asks John and Mike if they want to play again, return to main menu or exit. 2. Mike and John select to either play again, quit or go back to main menu 3. Depending on what the players select, BE will either reload the game, send one or both to main menu or exit for one or both. In this instance Mike, won and John wants a rematch, except Mike is a sore winner so he chooses to exit the game. BE will then exit Mike's game, inform John, Mike has exited, and ask John if he'd like to return to the main menu, exit or select another opponent.
Entry Condition	1. John has selected the Player vs. Player mode. 2. The game has detected Mike or John has had all their ships destroyed.
Exit Condition	1. Mike or John has selected to quit or go back to main menu.OR 2. Mike and John has selected to play again
Quality Requirements	1. BE should determine if there is a winner in, at the most, 500 milliseconds. 2. BE should be able to restart the game in, at the most, 10 seconds.

Fig 20(e): Use Case Scenario for winnerPvP Use Case

Use Case Name	winnerPvC
Participating Actors	Initiated by John:Player
Flow Of Events	1. John and the CPU have played a several rounds of Battlefield Extreme. BE will now check if there is a winner using ***detectWinner*** and if there is one prompts either a winner or loser screen to John and asks John if he'd like to play again, return to main menu or exit the game. 2. John will then make a decision as to whether he will play again, return to main menu or exit the game. 3. BE will then follow John's instructions of either playing again, exiting or returning to the main menu.
Entry Condition	1. John has selected the Player vs. AI mode. 2. The game has detected if either John or the AI has had all their ships destroyed.
Exit Condition	John selects either to play again, return to main menu or exit the game.
Quality Requirements	1. BE should determine if there is a winner in, at the most, 500 milliseconds. 2. BE should be able to restart the game in, at the most, 10 seconds.

Fig 20(f): Use Case Scenario for winnerPvC Use Case

21 Use Case Model

21.1 Use Cases

Use Case Name	***inviteFriendandPlay***
Participating Actors	Initiated by **Player(s)** **opponentPlayer(s)**
Flow of Events	1. The **Player(s)** selects a mode of Player vs. Player after selecting the game to play. 2. The BE responds by giving an option of selecting board size to the **Player(s)**. 3. The **Player(s)** selects the board size. 4. The BE gives the list of **opponentPlayer(s)**. 5. The **Player(s)** selects the **opponentPlayer(s)** from the list given by the BE using ***selectOpponentPlayer***. 6. The BE sends an invitation to join & play to the **opponentPlayer(s)**. 7. The **opponentPlayer(s)** receives the invitation to join and play the game. The **opponentPlayer(s)** can accept or decline the invitation request. 8. If the **OpponentPlayer(s)** accepts the invitation, then load the game. OPT. 9. If the **opponentPlayer(s)** declines the invitation, then select the opponent player again using ***selectOpponentPlayer***.
Entry Conditions	1. The **Player(s)** is logged into the social networking website account on which the game is hosted. 2. The **Player(s)** has greater than zero friends on the social networking website account. 3. The **Player(s)** has selected Player vs. Player as the mode of play.

Exit Conditions	1 The Player vs. Player game mode is loaded. OR 2 The **Player(s)** plays with AI in case he/she has no friends.
Quality Requirements	1. The **Player(s)** is acknowledged within 30 sec. about accept/decline of the invitation. 2. The game gets loaded within 20 seconds.

Fig. 21.1(a): Use Case for inviteFriendandPlay

Use Case Name	***selectPlaywithAI***
Participating Actors	Initiated by **Player(s)**
Flow of Events	1. The **Player(s)** selects a mode of Player vs. AI(Computer) after selecting the game to play. 2. The BE responds by giving an option of selecting board size to the **Player(s)**. 3. The **Player(s)** selects the board size. 4. The BE loads the game and the board is setup. 5. The **Player(s)** plays the game.
Entry Conditions	1. The **Player(s)** is logged into the social networking website account on which the game is hosted. 2. The **Player(s)** has selected Player vs. AI (Computer) as the mode of play.
Exit Conditions	1 The Player vs. Player game mode is loaded.
Quality Requirements	1. The game gets loaded within 20 seconds. 2. AI completes its turn within 10 seconds.

Fig. 21.1(b): Use Case for selectPlaywithAI

Use Case Name	selectAttack
Participating Actors	Initiated by Player(s) opponentPlayer(s)
Flow Of Events	1. The game loads and Player selects the ***playGame*** option. 2. BE loads the Player's board along with the Opponent's board (to see the attacks already made) 3. Player selects weapon type using ***selectWeapon***. 4. BE will display the weapon type, using specifications and strength. 5. Player selects an area, using ***selectAttackArea,*** according to weapon type 6. Player will get notified if HIT/MISS in the notification area and also Opponent will be notified about the same using ***updateBoard.***
Entry Condition	It is the Player's turn and Player selects "Attack" and the board is set-up
Exit Condition	The Player and the Opponent receives the notifications and the game boards should be updated using ***updateBoard.***
Quality Requirements	a) The notification to Player and Opponent should come quickly. b) The notification should be consistent.

Fig. 21.1(c): Use Case for selectAttack

Use Case Name	buyWeapon
Participating Actors	Initiated by Player(s)
Flow of Events	1. The Player selects to buy a weapon 2. BE responds by showing a list of weapons with their corresponding price, attributes, and specifications. 3. Player selects a weapon based on his/her choice or selects “cancel” button. 4. If(PlayerCash<WeaponCost),then BE denies the purchase and offers the Player to select another weapon. Else the corresponding weapon is purchased 5. The Player’s balance and Inventory is updated using ***updateBoard***. 6. The BE responds by giving an option to ***selectAttack***.
Entry Condition	It is the Player’s turn and Player selects “Buy”
Exit Condition	The weapon should be added into the inventory of the Player and cash balance is updated.
Quality Requirements	-

Fig. 21.1(d): Use Case for buyWeapon

21.2 Use Case Diagrams

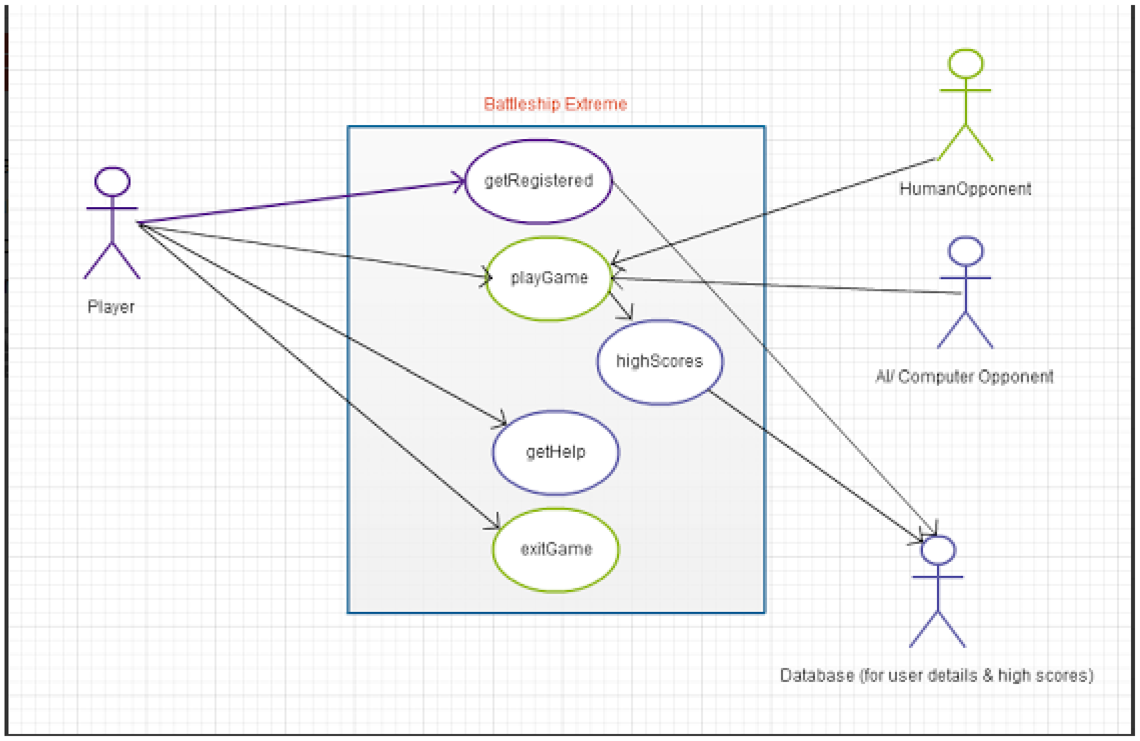


Fig. 21.2(a) : High Level Use case Diagram for Battleship Extreme

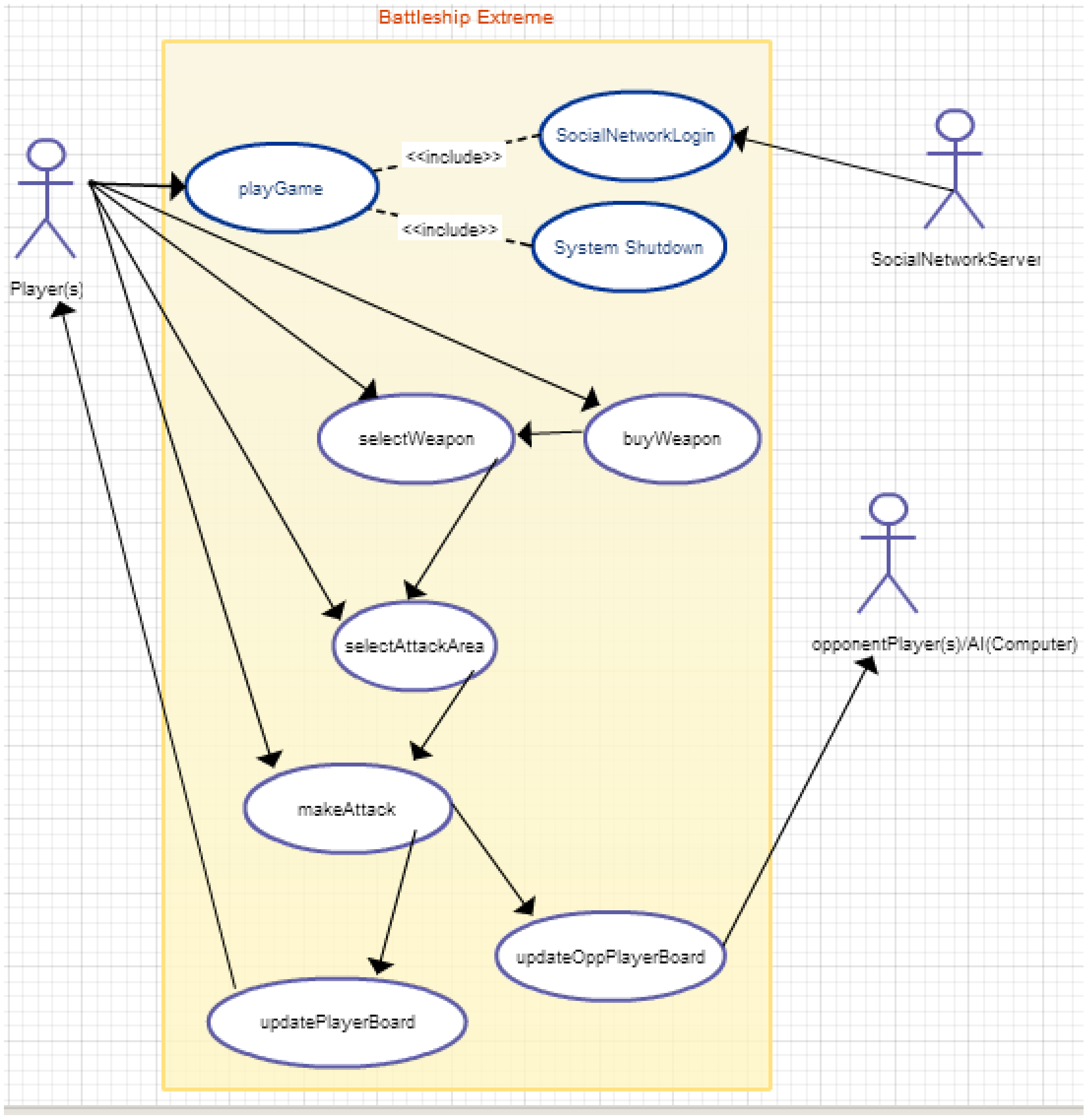


Fig. 21.2(b): Use case Diagram for playGame

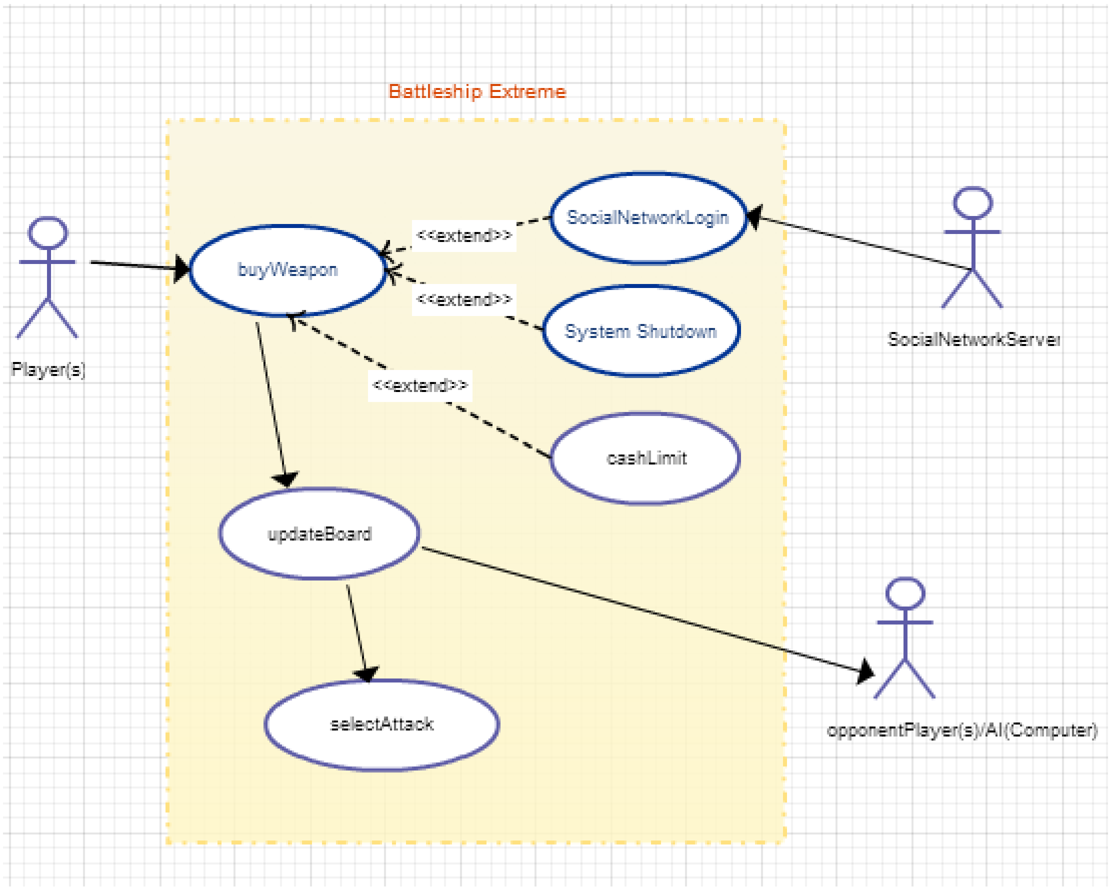


Fig. 21.2(c): Use case Diagram for buyWeapon

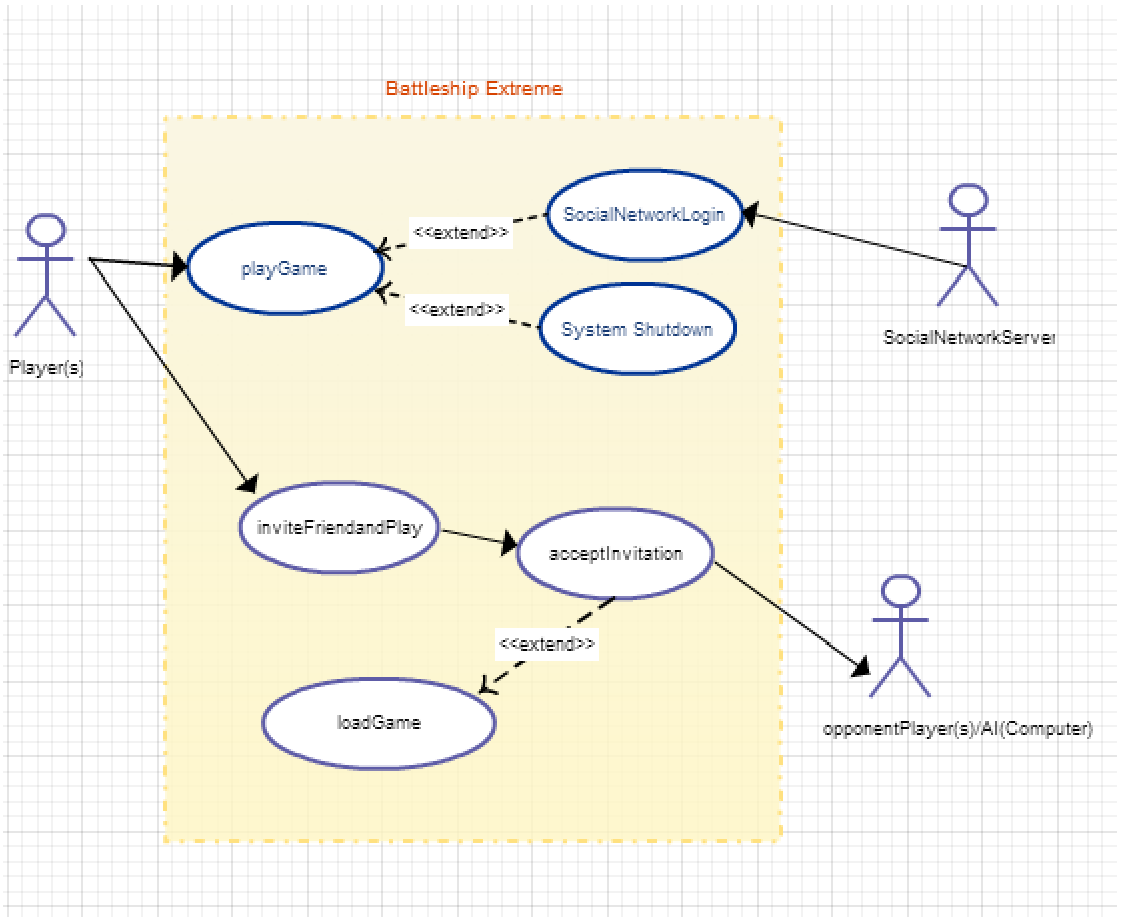
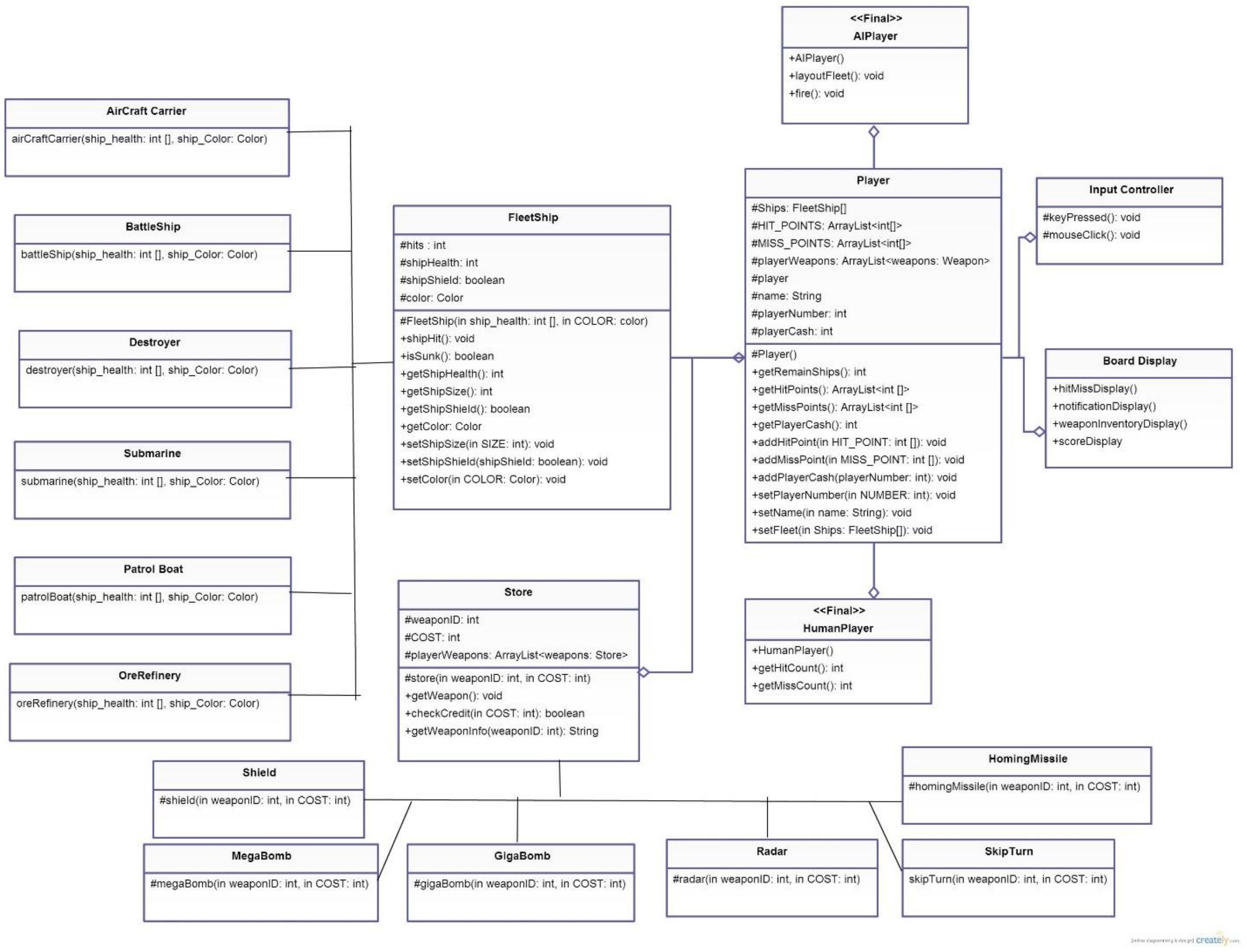


Fig 21.2(d): Use case Diagram for inviteFriendandPlay

22 Object Model



<<Final>>
AIPlayer
+AIPlayer()
+layoutFleet(): void
+fire(): void
AirCraft Carrier
airCraftCarrier(ship_health: int [], ship_Color: Color)
BattleShip
battleShip(ship_health: int [], ship_Color: Color)
Destroyer
destroyer(ship_health: int [], ship_Color: Color)
Submarine
submarine(ship_health: int [], ship_Color: Color)
Patrol Boat
patrolBoat(ship_health: int [], ship_Color: Color)
OreRefinery
oreRefinery(ship_health: int [], ship_Color: Color)
FleetShip
#hits : int
#shipHealth: int
#shipShield: boolean
#color: Color
#FleetShip(in ship_health: int [], in COLOR: color)
+shipHit(): void
+isSunk(): boolean
+getShipHealth(): int
+getShipSize(): int
+getShipShield(): boolean
+getColor: Color
+setShipSize(in SIZE: int): void
+setShipShield(shipShield: boolean): void
+setColor(in COLOR: Color): void
Player
#Ships: FleetShip[]
#HIT_POINTS: ArrayList<int[]>
#MISS_POINTS: ArrayList<int[]>
#playerWeapons: ArrayList<weapons: Weapon>
#player
#name: String
#playerNumber: int
#playerCash: int
#Player()
+getRemainShips(): int
+getHitPoints(): ArrayList<int []>
+getMissPoints(): ArrayList<int []>
+getPlayerCash(): int
+addHitPoint(in HIT_POINT: int []): void
+addMissPoint(in MISS_POINT: int []): void
+addPlayerCash(playerNumber: int): void
+setPlayerNumber(in NUMBER: int): void
+setName(in name: String): void
+setFleet(in Ships: FleetShip[]): void
Input Controller
#keyPressed(): void
#mouseClick(): void
Board Display
+hitMissDisplay()
+notificationDisplay()
+weaponInventoryDisplay()
+scoreDisplay
Store
#weaponID: int
#COST: int
#playerWeapons: ArrayList<weapons: Store>
#store(in weaponID: int, in COST: int)
+getWeapon(): void
+checkCredit(in COST: int): boolean
+getWeaponInfo(weaponID: int): String
<<Final>>
HumanPlayer
+HumanPlayer()
+getHitCount(): int
+getMissCount(): int
Shield
#shield(in weaponID: int, in COST: int)
HomingMissile
#homingMissile(in weaponID: int, in COST: int)
MegaBomb
#megaBomb(in weaponID: int, in COST: int)
GigaBomb
#gigaBomb(in weaponID: int, in COST: int)
Radar
#radar(in weaponID: int, in COST: int)
SkipTurn
skipTurn(in weaponID: int, in COST: int)
creately

Fig 22: Detailed Object Model for Battleship Extreme

23 Data Dictionary

- *Board Display*: Main in-game user interface. Displays the board with the hit or miss icon, the user's weapon inventory, the user's score and the user's ships.

- *Input Controller*: "Listener" which checks for mouse clicks and keystrokes.

- Fleetships: The main ship class which the lesser ships inherit attributes from.
  - Air-Craft Carrier: Lesser ship which inherits from Fleetships. Health size of five.
  - Battleship: Lesser ship which inherits from Fleetships. Health size of four.
  - Destroyer: Lesser ship which inherits from Fleetships. Health size of three.
  - Submarine:Lesser ship which inherits from Fleetships. Health size of three.
  - Patrol Boat: Lesser ship which inherits from Fleetships. Health size of two.
  - Ore Refinery: Lesser "ship" which inherits from Fleetships, purchasable from the
  - store and generates money. Health size of one.

- *Store*: The interface which will enable the player to purchase attack and defense weaponry. Will contain the weapons available, the cost of each weapon and information on what the weapon actually does.

- *Skip Turn*: When purchased this will enable the player to "skip" the opponents next turn enabling the player to make multiple moves in one turn.

- *Ore Refineries*: These will act as money making engine which generates a continuous income for a player on each turn he/she takes. It will occupy one cell space on the board.

- *Notification Area*: This will notify the player about the opponents move. The notification area will help the players to strategize their future moves.

- *Giga Bombs*: With the Giga bombs, user will be able to select one grid space, and if it is a HIT, the whole ship will sink.

- *Mega Bombs*: With Mega bombs, it will explode within a 3x3 grid space and if any ship is in range will hit it. Player will be notified if it is a HIT or a MISS.

- *Homing Missiles*: Homing missiles will be used by selecting a 3x3 grid space and will hit only one section of a ship if it is in range. With the homing missile, the player will only be notified if it is a HIT or a full MISS.

- *Radar*: The radar will also be a 3x3 grid space in which it will notify the player of the ships in that area.

- *HIT*: Player makes a hit on his move, destroys a part of the opponent's ship.
- *MISS*: Player misses the attack on his move. All the ships are unharmed on this move by the player.

24 Class Diagram

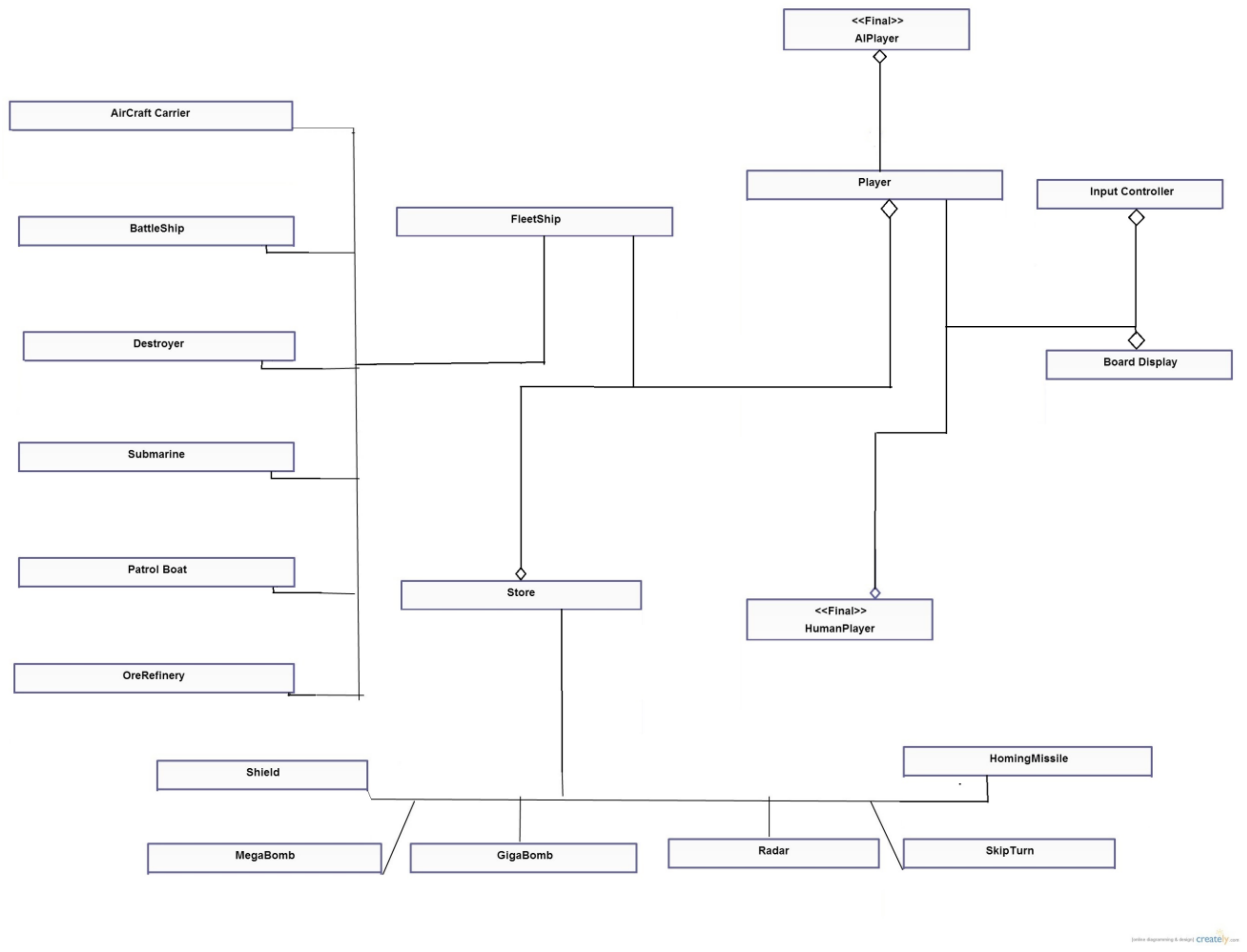


Fig. 24: Class Diagram for Battleship Extreme

25 Dynamic Model

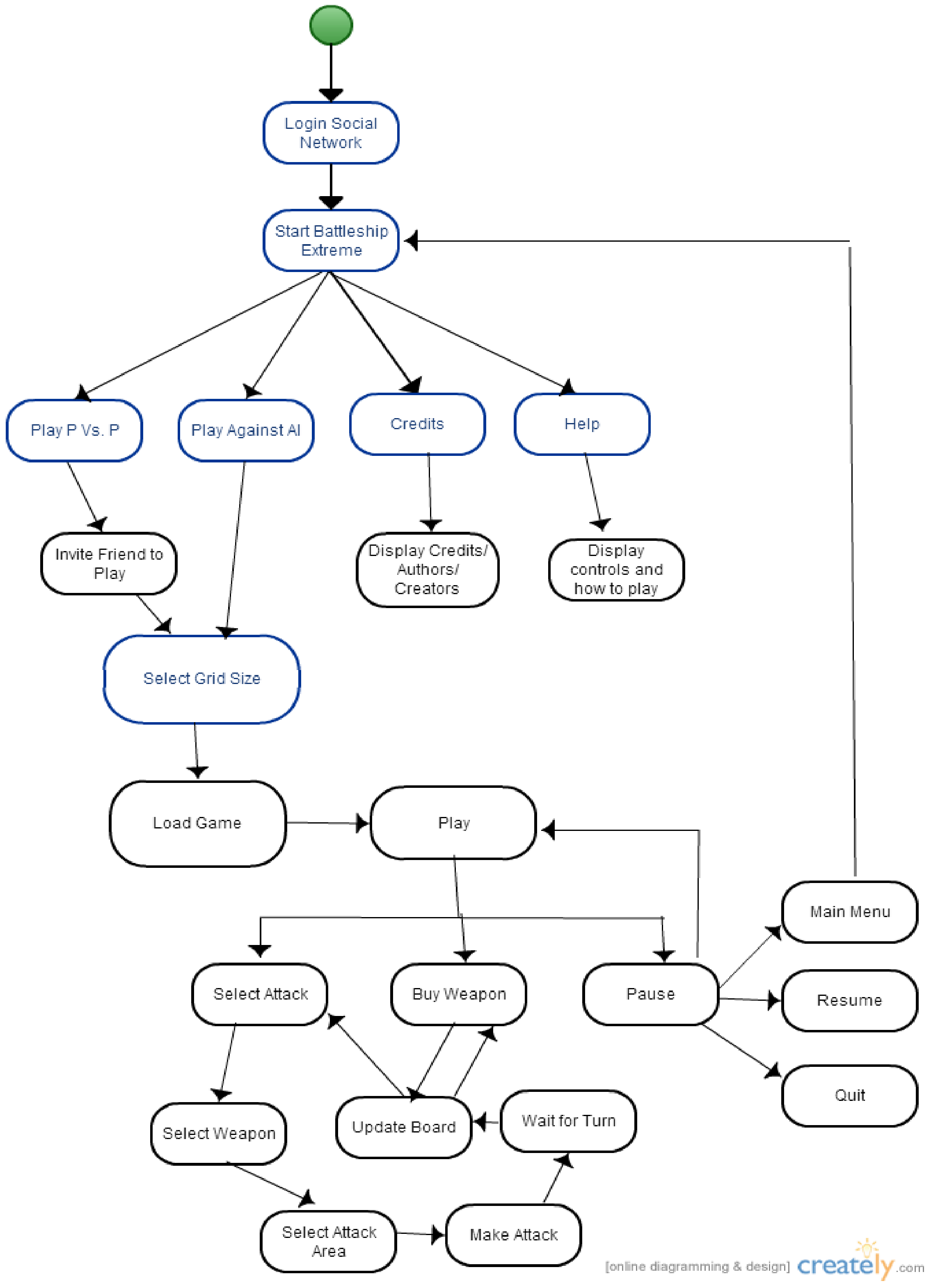


Fig 25(a): Activity Diagram for Battleship Extreme

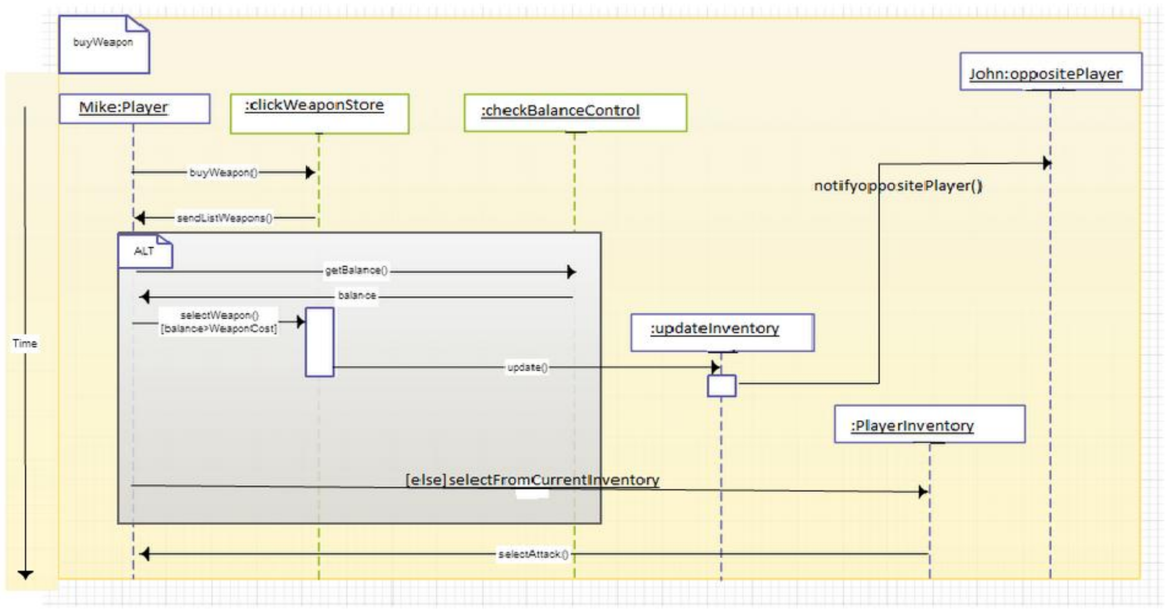


Fig 25(b) : Sequence Diagram for buyWeapon Use case

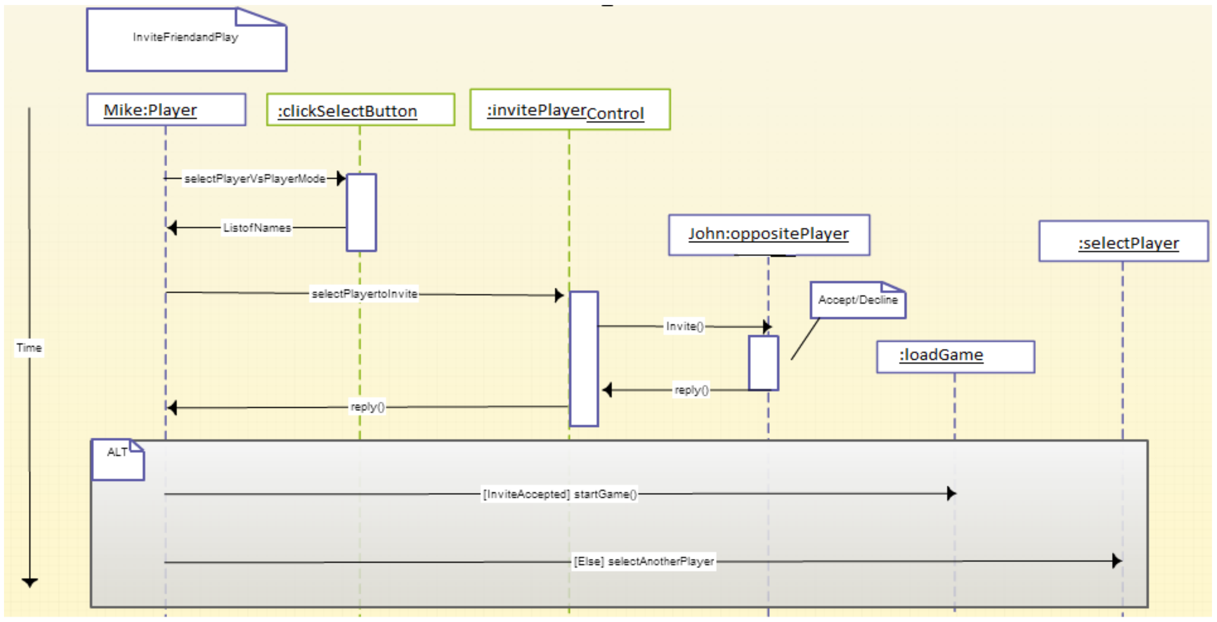


Fig 25(c) : Sequence Diagram for inviteFriendandPlay Use case

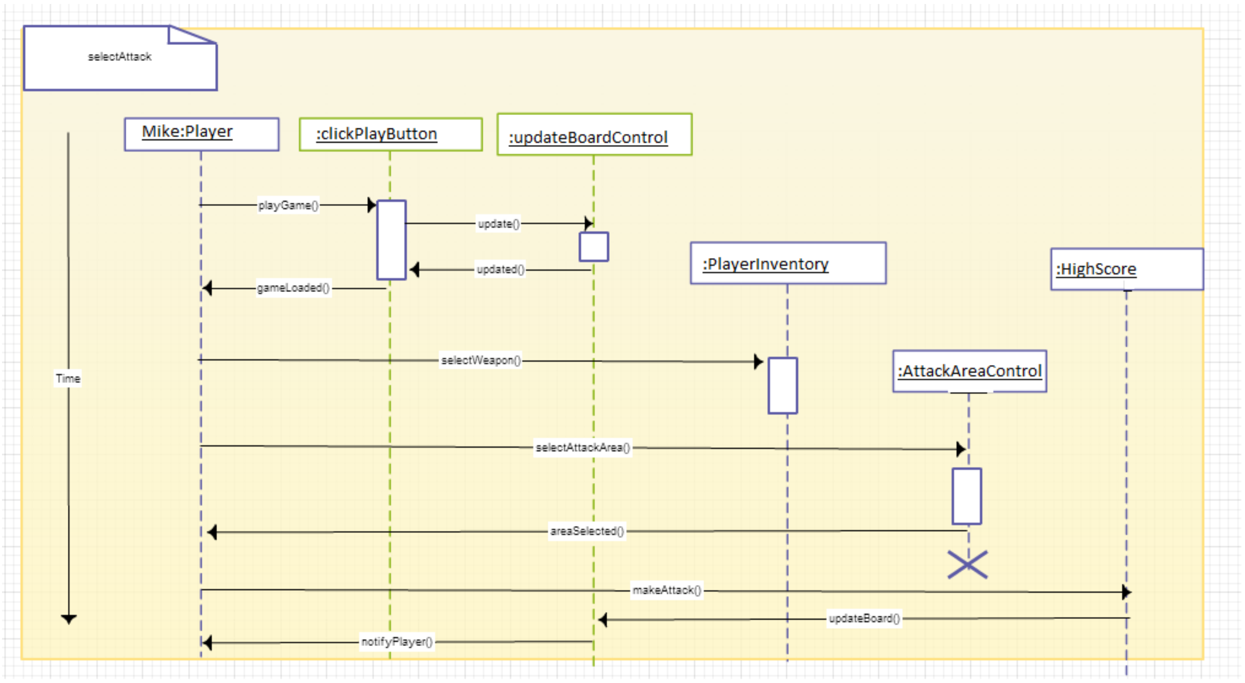


Fig 25(d) : Sequence Diagram for selectAttack Use case

26 User Interface

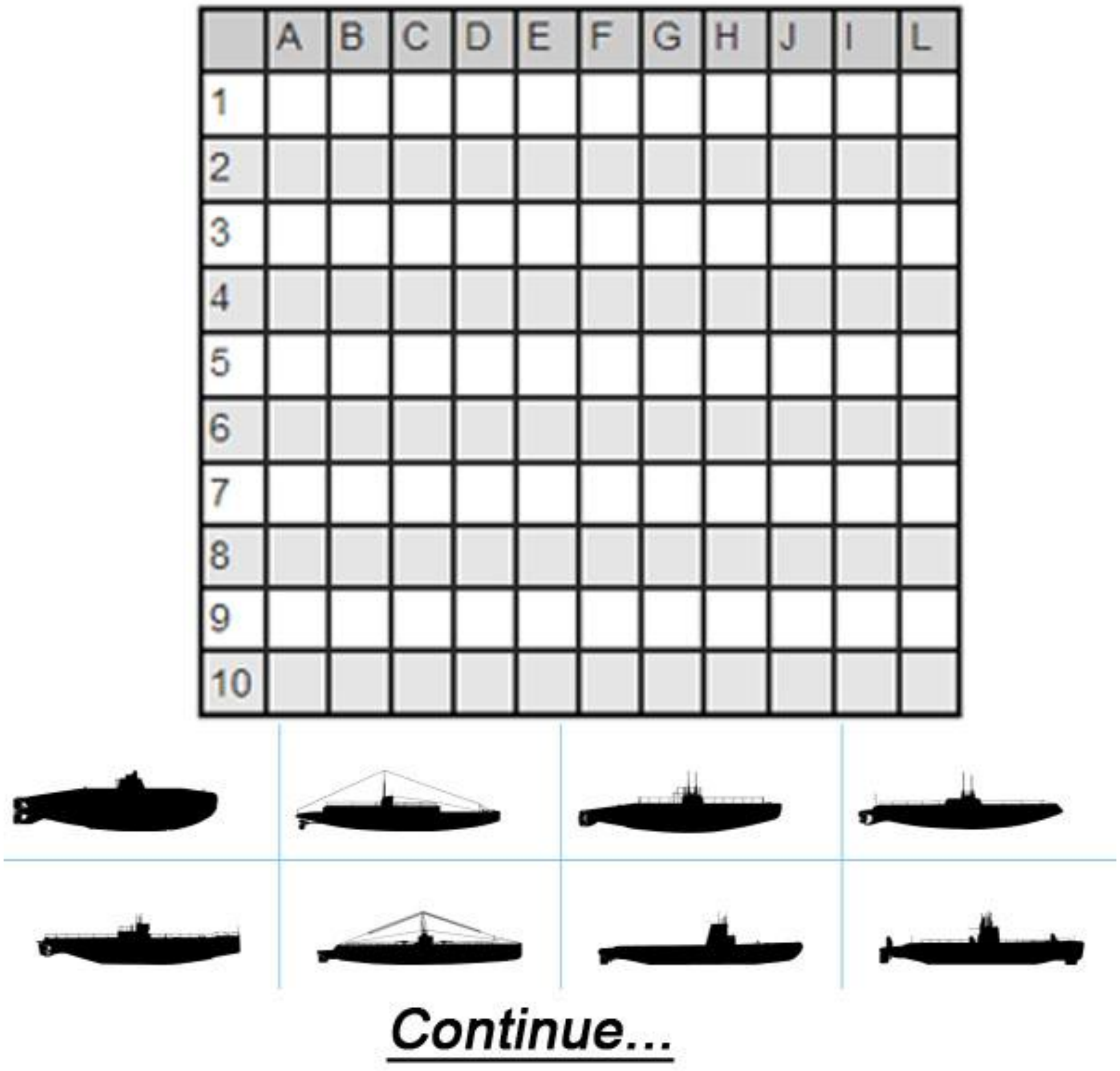


Fig. 26(a): User Interface for positionShips

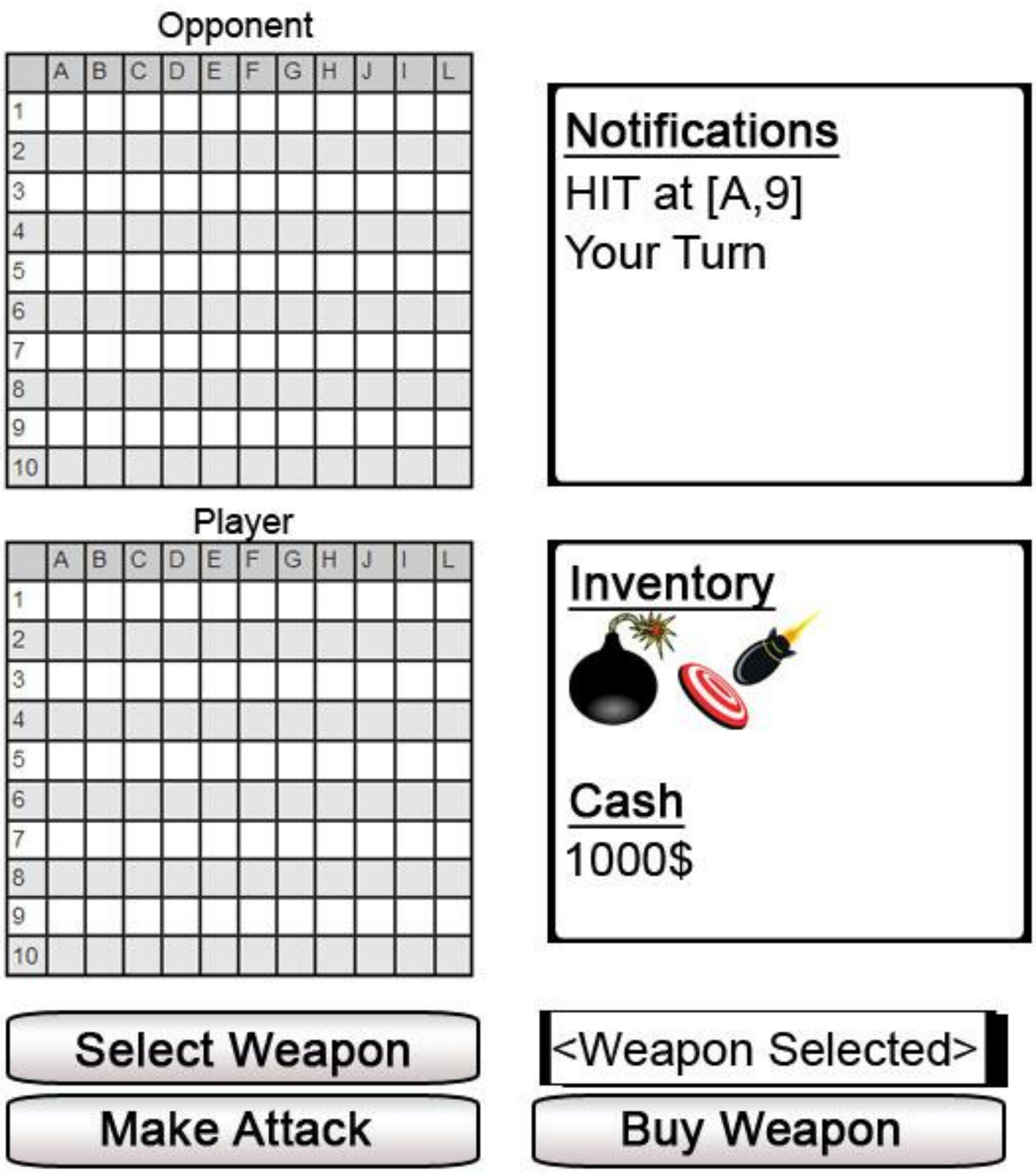


Fig.26(b): User Interface for boardDisplay

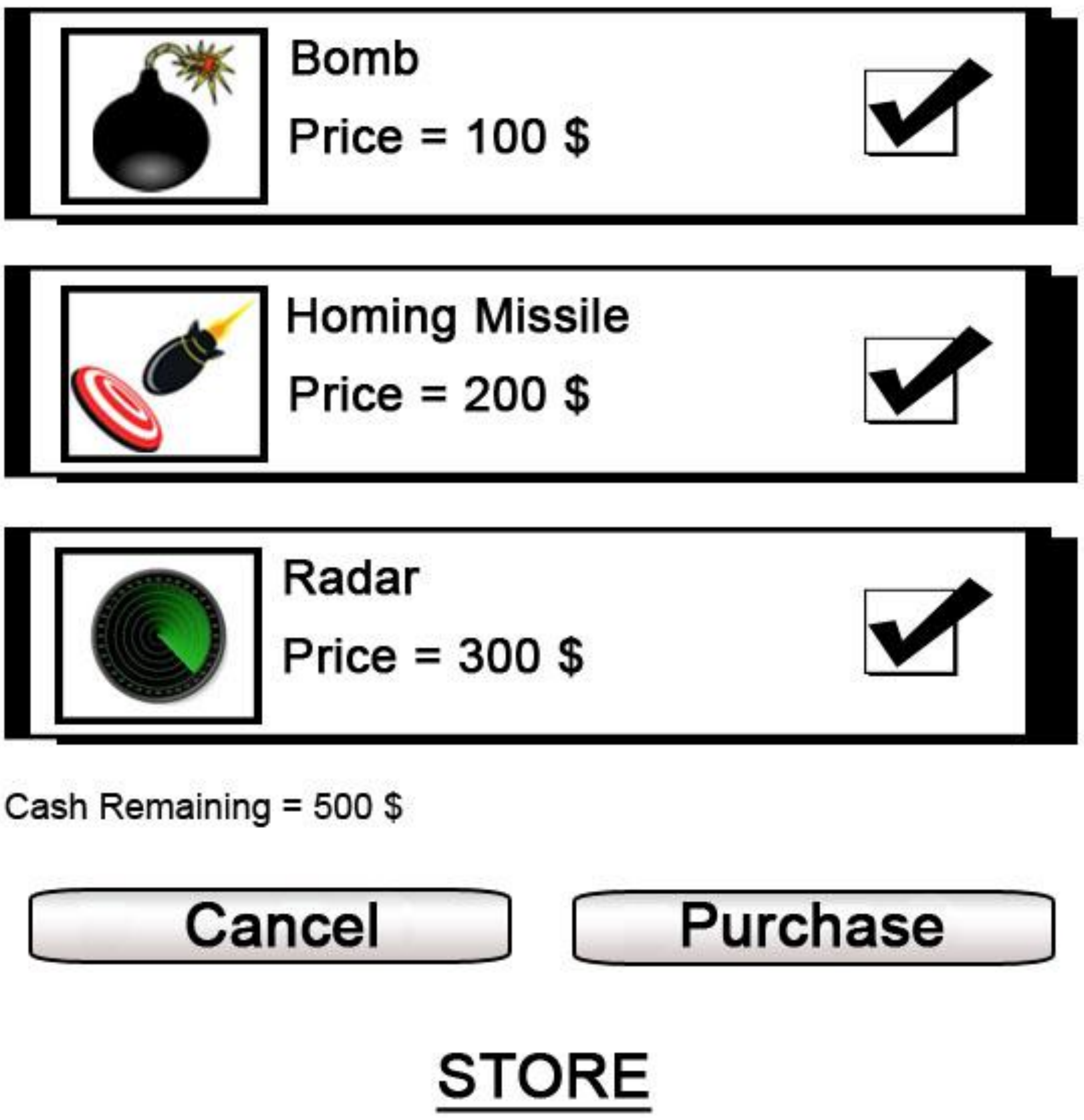


Fig.26(c): User Interface for Weapon Store

IV. Design

27 System Design

During system design, the system is to be transformed from the analysis model into a system design model. We have defined the design goals of the game and decomposed the system into smaller subsystems.

27.1 Purpose of the System

Battleship Extreme is a system aiming to entertain users with a well-designed gameplay. The game is simple, fast and fun. Instead of producing a high-end photo realistic 3D game, which need high hardware performance, Battleship Extreme is a small game that can be played in a couple of minutes and provide a high fun-factor by simple rules. There are multiple examples like Tetris or Minesweeper that prove the acceptance and the success of simple games.

27.2 Design Goals

The design goals represent the desired qualities of Battleship Extreme and provide a consistent set of criteria that must be considered when making design decisions. Our main purpose is to develop robust, maintainable, well-designed and reusable software with Object-Oriented analysis and design. We are determined to define and visualize each and every perspective of the system explicitly in order to completely materialize our Object-Oriented approach. Next, we also pay attention to how to diminish the influence and impact of alterations, how to keep the elements of our design understandable, manageable, and focused and who is when behavior differs by type.

The design goals identified in details are as follows:

- ***Adaptability***: Java® is one of the few programming languages which provide cross-platform portability. This attribute of Java® enables our system to work in all JRE installed platforms; therefore, user will not have to worry about the operating system requirements. In order to fulfill this adaptability feature, we preferred to program the game in java by sacrificing some of the performance advantages of other programming languages such as C or C++.
- ***Efficiency***: The system is going to be responsive and able to run with high performance. The game will run at least 40 fps in order to provide smoothness in the movements of the game objects. This is the most important design goal because performance of the game has a crucial role for users' excitement. In order to reach the optimum game performance, rather than trying to minimize the memory usage, we allocated memory for each individual objects so that they will be responsible for their own tasks.
- ***Reliability***: System will be bug-free and consistent in the boundary conditions. The system should not crash with unexpected inputs. To achieve this goal, the testing procedures will continue simultaneously with each stage of the development. Besides,

boundary conditions will be evaluated very carefully not to miss any unconsidered situation which may crash the system.

- ***Usability***: Easiness in the usage is an important design goal in such games in terms of user's comfort. It makes the game more friendly and attractive. Therefore, the system will be designed such that user can easily interact with the system without any prior knowledge. However, user-friendliness of the system does not imply making the gameplay easier, which might make the player bored sooner than expected.
- ***Extensibility***: Object oriented architecture of the game enables system customizations without causing any bugs during modifications. The game should be able to run with at least 1000 players trying to play the game concurrently at a point in time. The game is expected to capture more users as the time progresses. Therefore, this design architecture minimizes the possibility to cause malfunctioning in other classes.

28 Current Software Architecture

As the Battleship Extreme game is a Greenfield engineering project we have no current software to replace.

29 Proposed Software Architecture

29.1 Overview

Three-tier architecture is a client–server architecture in which the user interface, functional process logic, computer data storage and data access are developed and maintained as independent modules. The three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. Battleship Extreme is a web-application game comprising of 3-tiers like presentation tier, application logic tier and data tier. Hence, three-tier architecture is the most suitable architecture to follow in respect to our game 'Battleship Extreme'.

- *Presentation tier*
  The topmost level of the game is a user interface. The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents. It communicates with game-server by outputting results to the browser/client tier. In simple terms, it is a layer which users can access directly such as a web page.
- *Application logic tier*
  The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application's functionality by performing detailed processing. It moves and processes data between layers above and below it.
- *Data tier*

This tier consists of database servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

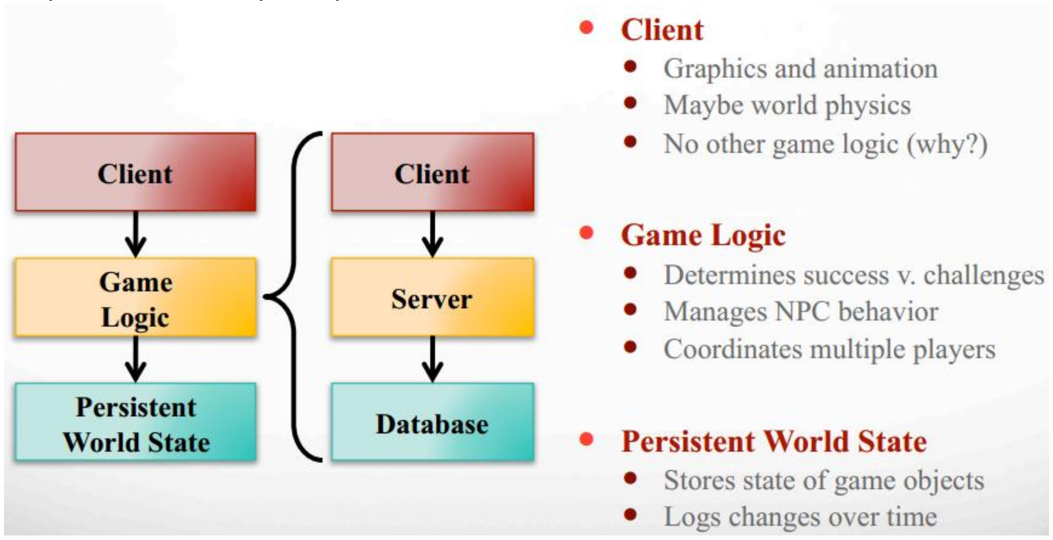


Fig. 29(a): 3-Tier Architectural Style

29.2 Subsystem Decomposition

During the subsystem decomposition of ‘Battleship Extreme’ we divide the system into smaller subsystems with a strong coherence. The different subsystems should have a loose coupling.

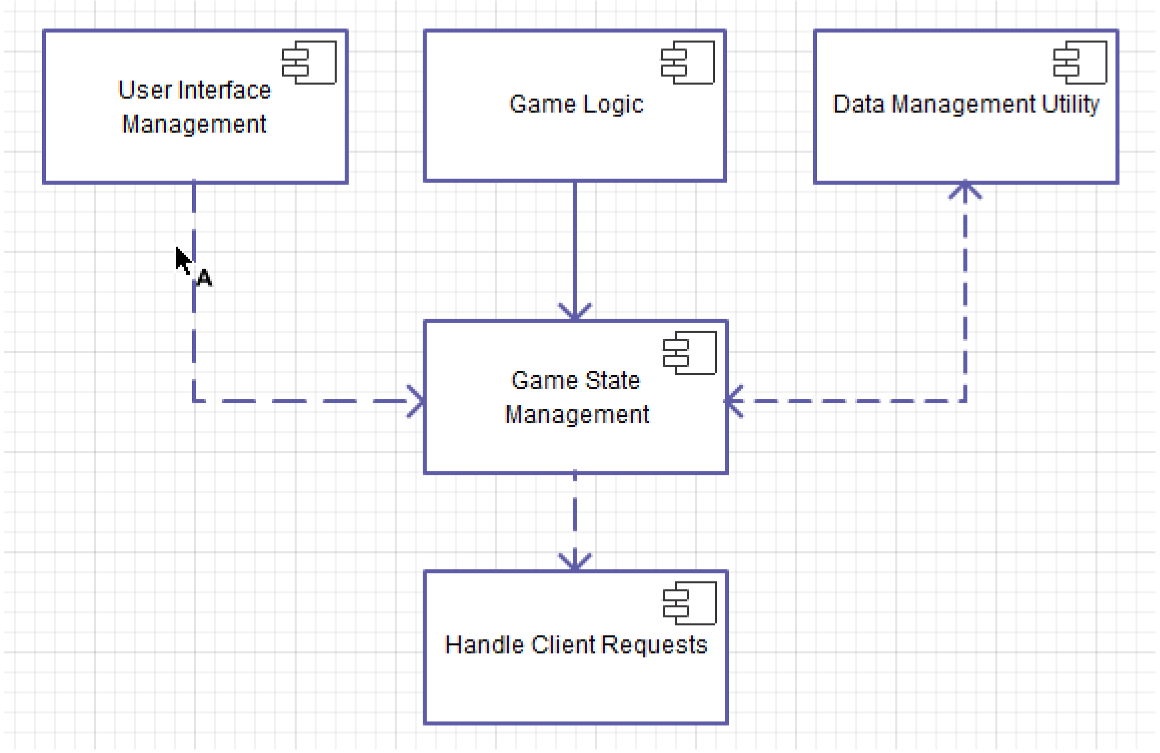


Fig. 29(b): Subsystem decomposition for Battleship Extreme

29.3 Hardware/ software mapping

Battleship Extreme is built as a server-end application and clients communicates to it using their respective web browser. The Battleship Extreme must run on all the widely used operating systems as long as the system has a java enabled web browser. The game must support modern operating systems like Windows 7/8/XP or Unix-like OSs (Ubuntu, Solaris, Fedora, Debian etc.) or MacOS. The game must be able to run in an incessant manner with web browsers like Internet Explorer, Mozilla Firefox, Google Chrome and Apple Safari. The game shall easily be able to run on any combination of OS and web browser as long has the web browser is java enabled. The following UML deployment diagram illustrates the hardware/software mapping for Battleship Extreme.

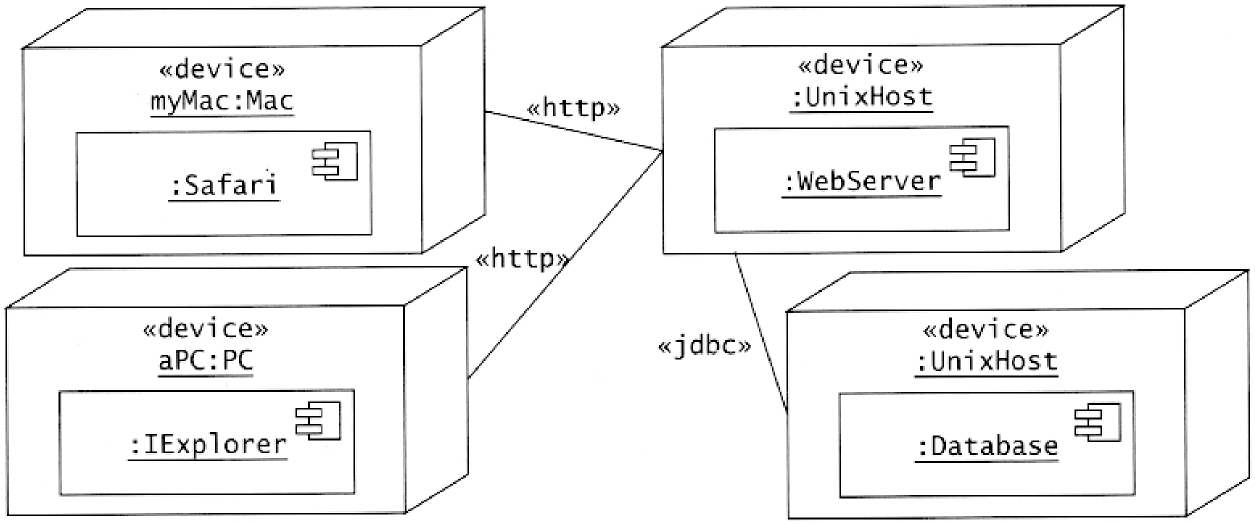


Fig. 29(c): Hardware/Software mapping for Battleship Extreme

29.4 Persistent Data management

The game keeps names and scores as high scorers stored on a database hosted on different web server to ensure security. Some images and music files are also used at different parts of the game in order to provide better experience on gaming. These files are read from the file system located on the same web server as of game. This will fasten up the loading all the necessary files when the game logic requires, since all files on the same server.

:playerName	:score

Fig. 29(d): Persistent Data Management for Battleship Extreme

29.5 Access control and security

We model access by determining which objects are shared among actors, and by defining how actors can control access. Since 'Battleship Extreme' is a social-network game, the users will be authenticated using the authentication API provided by that social-network. Each player will have read access to all the players' scores, however he/she will be having write access for his/her own score only during the gameplay. Game administrator will have unlimited access to system data and to other user's data.

Role	*Read Access Scores*	*Write Access Scores*
Player	ALLOWED	ALLOWED for own; NOT ALLOWED for others
Game Admin	ALLOWED	ALLOWED

Fig. 29(e): Access Control Matrix for Battleship Extreme

29.6 Global software control

In object-oriented systems, sequencing actions includes deciding which operations should be executed and in which order. Since Battleship Extreme has to deal with many clients simultaneously, we prefer to use threads to communicate with each client separately. Whenever a new client connects, the server creates a new thread to handle events for that client. Different socket descriptors will enable communication among all clients. There will be separate threads used to manage the gameplay of the game like maintaining states of every player, managing server resources etc.

29.7 Boundary conditions

The boundary conditions of the system specifies how the system is started, initialized, and shut down and we need to define how we deal with major failures such as data corruption and network outages.

Startup and Shutdown: Each game instance is initialized and started as soon as the player connects to the game server on the social networking website using the web browser. The player can terminate that game instance by choosing the Exit option from the menu on the game web-app or closing the session forcibly by clicking the cross button on the top right corner of the web browser.

Exception Handling: The Battleship Extreme game is network-dependent. Hence, we have to take care of network outages. So, in case of any such failure, the system should be able to save the state of the game for a particular player after every move. Later on when the network resumes, the player should be able to restore to its previous state of game from where he left. There are exceptions handling mechanisms on java to handle such unexpectedly closed sockets. We need to handle these exceptions carefully, since our whole game is network dependent for most of its resources.

Subsystem responsible for database management should be able to detect whether or not it was properly shut down and should be able to perform consistency checks and repair corrupted data, as necessary.

30 Subsystem services

FleetShip is the constructor for the different types of ships with different characteristics. Ex Aircraft Carrier, BattleShip, Destroyer, Submarine, Patrol Boat, OreRefinery.

Store contains possible weapons the player can purchase. Each weapon has different cost and characteristics. Ex Sheild, Megabomb, Gigabomb, Radar, SkipTurn, HomingMissile. Note that non-weapons like Shield, Radar, and SkipTurn are still considered weapons.

Player constructs human player(s), and/or AIPlayer. Both inherit the same characteristics except the AIPlayer also inherits a built in decision maker.

Board Display handles output of the computer responses and Player actions.

Input Controller handles all the inputs from keyboard press and mouse events.

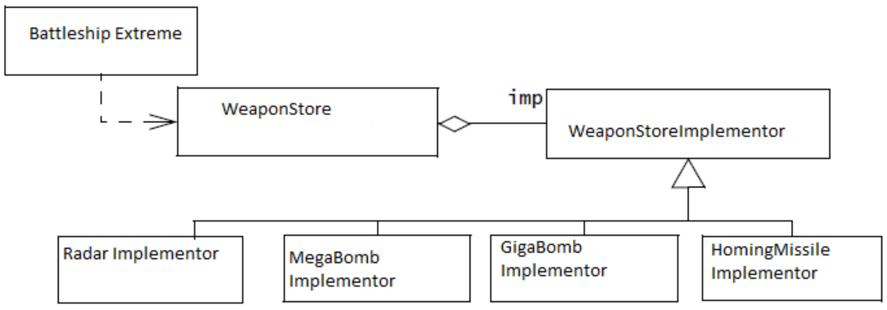


Fig.30(a): The ***Bridge Design Pattern*** in Battleship Extreme

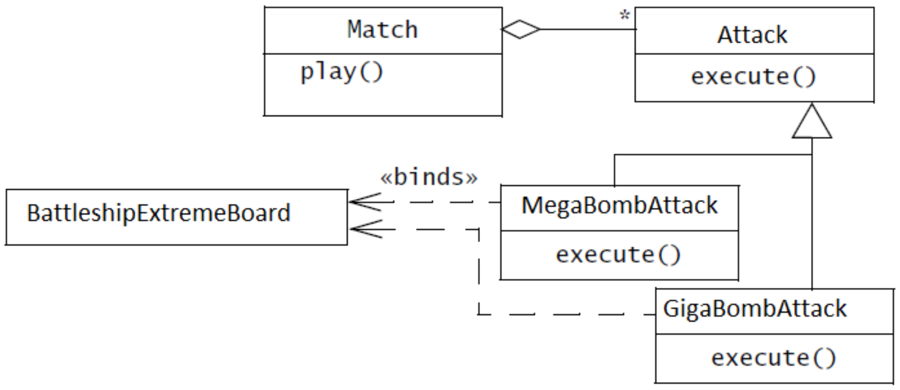


Fig.30(b): The ***Command Design Pattern*** in Battleship Extreme

31 Object Design

31.1 Object Design trade-offs

- *Efficiency vs Reusability*: Reusability is not the main concern for designing our system because we do not plan to integrate any of our classes to a different game or any other similar systems. Therefore, the classes are designed specifically for the tasks of our game so the code is not made more complex than necessary. This design approach fortified our most important design goal, which is efficiency, since there will not be any fancy "if statements" or type checking to enforce the reusability constraint.

- *Functionality vs Usability*: It is very important to have wide range of customers. Therefore, the game will have plain usage. The system should not be too complex to play. It means that the functionality of the system will be basic. Since the purpose of the system is entertaining the users, we focused on the usability of the system rather than making it functional more than necessary. The game has a simple interface and familiar instructions to play instead of complex menus and various features. Thus, the users can spend time enjoying the game rather than struggling to learn it.

- *Space vs Speed*: Space is notably inexpensive and as such is more expendable than speed. It is more important that the site load fast than that we take up less disk space. With that said, images and other date should be stored in formats that allow for the quickest possible processing regardless of size. This takes into consideration costly compression algorithms and post processing. Care should be taken, however, to minimize database to ensure that the datasets do not grow so large that they become cumbersome to search.

31.2 Interface Documentation guidelines

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can be helpful in understanding the code.

This section will give an insight about the Naming Convention. They are described individually in the table.

Identifier Type	**Rules for Naming**	**Examples**
Packages	The prefix of a unique package name is always written in capital and all lowercase ASCII letters except abbreviations like UI is User Interface so as not to make lengthy package name.	package BoardDisplay; package

		InputController; package fleetships
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. We kept our class names simple and descriptive. Nonetheless, we used whole words and avoided acronyms and abbreviations as far as possible (unless the abbreviation is much more widely used than the long form, such as DB for database and UI for User Interface). Some classes' names also have numeral as it makes us to feel the step of execution.	class Player; class HumanPlayer; class Bombs;
Interfaces	Interface names should be capitalized like class names.	interface MainBoard; interface SelectGameType;
Methods	Our Methods are verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	insert(); getHitPoints(); checkCredit();
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names has no start with underscore _ or dollar sign $ characters, even though both are allowed. Variable names are short yet meaningful. The choice of a variable name is mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names are avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	int i; char c;
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_PLAYERS = 1;

Fig. 31.2: Interface Documentation Guidelines for Battleship Extreme

31.3 Packages

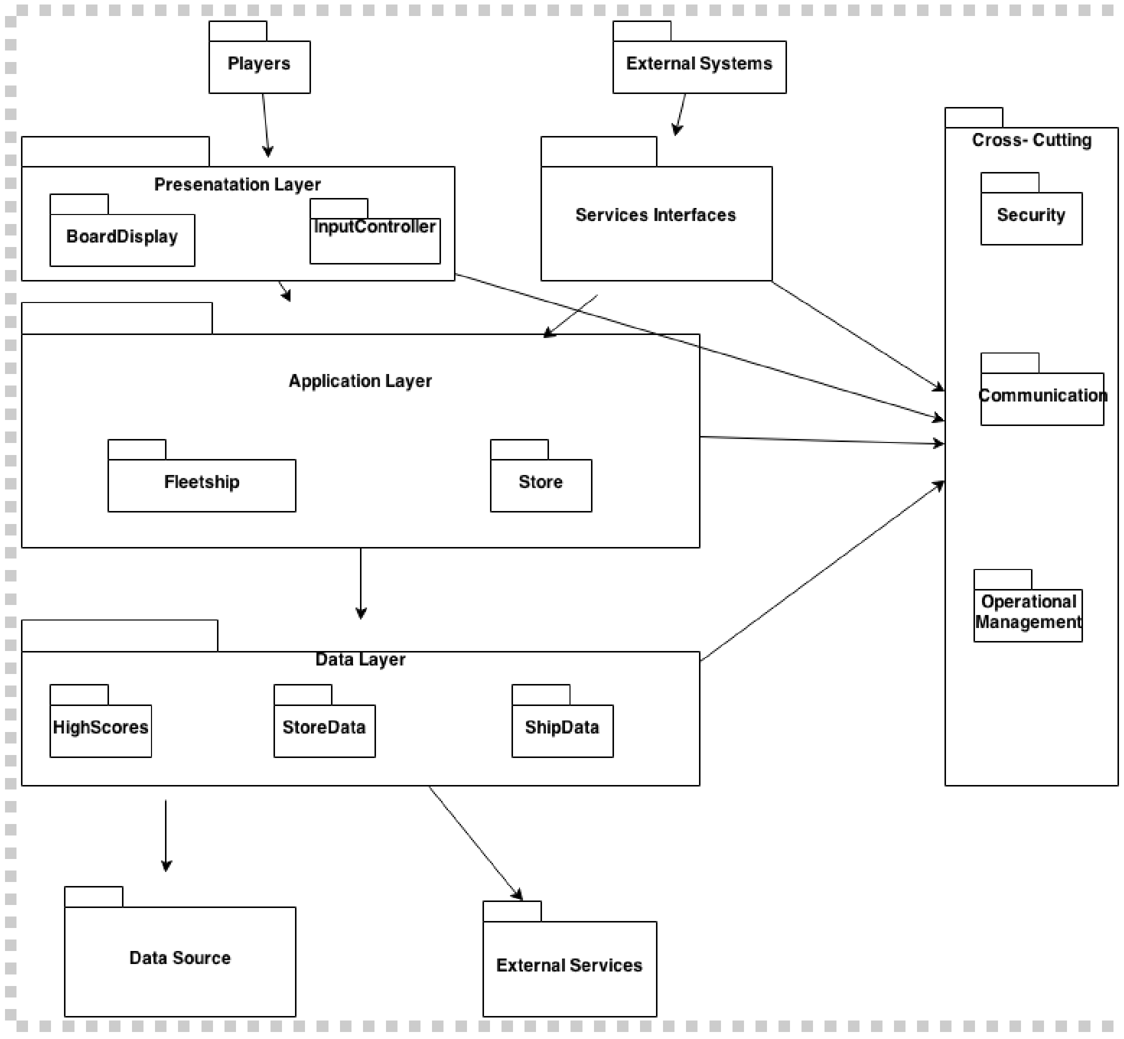


Fig. 31.3: UML Package diagram for Batleship Extreme

31.4 Class Interfaces

- **Class Mainboard** – This is the main class of the application. It declares all the user interface classes
- **Class SelectGameType** – This class allows the user to select which type of game to play "Player vs. Player" or "Player vs. AI".
- **Class BoardLoader** – This class handles loading in the board with Board class, Notification class, Inventory Class, and Cash class.
- **Class Board** – This class handles the Player's board and the Player's HIT/MISS board right above it.
- **Class Notification** – This class handles all the notifications ("HIT, MISS, POINTS EARNED").
- **Class Inventory** – This class handles all the Player's objects player can use to attack or defend.
- **Class CashBalance** – This class handles the currency aspect of the game when purchasing weapons and earning cash through Ore Refinery or HITS.
- **Class Store** – This class handles weapons in the store, cost and description of each item, and checking and manipulating Player's cash balance.
- **Class Ships** – This class handles all the different types of ships and it's attributes that are on the Player's boards
- **Class Weapons** – This class handles all the different weapons and it's attributes that can be purchased in the store and loaded into inventory.

V. Test Plans

32 Features to be tested

During Testing, faults are detected using technique that tries to create failures or erroneous states in a planned way. Taking the requirements as a benchmark, the developed software shall be validated, i.e. the process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610]

Battleship Extreme comprises of umpteen numbers of features which should be meticulously tested so as to guarantee a quality game. The following are the features to be tested:

- Battleship Extreme should be available as a social network game and should run on any modern day browser through social networking website like Facebook, MySpace etc.
- Battleship Extreme should start in the current browser tab and asks the users to enter their information if it is they are using it for the first time.
- The users should be able to enter their information using keyboard and control/play the game using mouse.
- Battleship Extreme should display a main screen showing 4 options: 1. Play Game, 2. High Scores, 3. Help, and 4. Quit.
- Battleship extreme should support two modes of play: Player vs. Computer and Player vs. Player.
- Battleship Extreme should display High Scores in a tabulated format with three columns: 1. Player Name, 2. High Score, and 3. Joining Date.
- Battleship Extreme should allow Player to quit the game at any time using keyboard key “Esc”.
- Battleship Extreme should allow Player to Pause/Resume the game at any point in time and game state should be saved.
- The screen displayed while playing the game should not exceed the size of the browser window.

- The display screen GUI should show the Player’s battlefield, Opponent Player’s battlefield, a notification for tracking the opponent Player’s moves, inventory and buttons for selecting attack and defense mechanisms.
- Battleship Extreme should support proper computer AI for Player vs. Computer mode.
- Battleship Extreme should support virtual currency. The virtual currency should be displayed on the display screen GUI.
- Battleship Extreme should support the proper functioning of Ore Refineries.
- Battleship Extreme should support the Attack and Defense Mechanisms.
- Battleship Extreme should show a win screen when a Player wins an instance of the game and the Player’s scores database should be updated.
- Battleship Extreme should support a proper easy-to-learn tutorial about the game if a Player selects “Help” on the main screen.
- Battleship Extreme should be highly responsive.
- The background images/graphics should support the “War-like” Theme of the game and the Player should also feel that he/she is playing on board.
- The background should have music and sound effects for critical moves.
- Battleship Extreme should be reliable and portable.

33 Pass/Fail Criteria

Below are the listed criteria at **Unit Testing** level to be fulfilled in order for the test cases to the deemed as “PASS”

- All test cases should be passed at least 95% of the times.
- Code coverage should be at least 90% which should be checked using Java code coverage tools.
- Each component should follow standard coding guidelines with proper comments.

At **System Testing** level, following are the list of criteria to be fulfilled:

- All lower test plans like unit testing and integration testing should be complete.

- At least 97% of the test cases should have passed before system testing.
- All Test cases with critical functionality must pass.
- All high and medium severity defects must be fixed.

34 Approach

34.1 Testing Levels

We plan to conduct the following type of testing in order to ensure the quality of our game:

1. **Unit testing** which tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model.

2. **Integration testing** is the activity of finding faults by testing individual components in combination.

   The size of the Battleship Extreme system is relatively small therefore; following an approach of **Big Bang Integration Testing** will be the best.

3. **System testing** tests all the components together, seen as a single system to identify faults with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:

- **Functional testing** tests the requirements from the RAD and the user manual.
- **Performance testing** checks the nonfunctional requirements and additional design goals from the SDD. Functional and performance testing are done by developers.
- **Acceptance testing** checks the system against the project agreement and is done by the client, if necessary, with help by the developers.

  The testing should start as soon as or possibly even before a single component is developed. Testing early will uncover defects early which will prevent developers to reinvent the wheel at a later stage.

34.2 Test Tools

The developers/testers should test the functions mentioned in section 32 manually considering the logic of the function as a black box and comparing the expected and actual output. On the other hand, the developers should conduct white box testing during the development of every function by meticulously engulfing the logic of the function. White box testing should be conducted manually encompassing the **Path testing, Equivalence testing,** and **Boundary testing.** As an alternate, and for more robust kind of testing, the developers can use the **JUnit Framework** and develop the test scripts using Java programming language and check the code coverage using **EclEmma Java Code Coverage tool** in Eclipse.

For **Manual Testing**, tools should be available are: Microsoft Excel for creating test cases and developing a test report in the form of table. Microsoft Word should be available to compile the test results at all the levels of testing and prepare a final test report.

34.3 Meetings

The team should meet and discuss at the end of every two weeks and fix the defects discovered by that time instead of going ahead without fixing the current defects because these unfixed defects can cause some serious issues at the end of the sprint/release. The developer of the component under test should note down/document the issues carefully and the team should discuss a solution for the impediments.

35 Suspension and Resumption

The testing of the system should stop when:

- The number of failed test cases > 20 at any point of time, at which point of time, the team should concentrate of fixing the defects instead of continuing on development and testing, and
- If the server of the social network website on which the Battleship Extreme will be hosted goes down.

36 Testing Deliverables

The following items should be delivered at the end of testing:

- Test Cases for ***Unit Testing, Integration Testing, System Testing*** *and* ***User Acceptance Testing***
- Test Report
- Error logs
- Test Plans(Unit Test Plan, Integration Test Plan, System Test Plan)
- Screen Prototypes

37 Test Cases

Test Cases for Functional Requirements:

ID	TEST DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	RESULT
TC_01	There should be proper human input through keyboard while the user enters their information. **PRECONDITION:** Users should be logged into the social network account and opted to play Battleship Extreme.	The information shall be entered through keyboard and main screen shall be displayed.		
TC_02	The user should be able to control the game using mouse. **PRECONDITION**: Users should be logged into the social network account and opted to play Battleship Extreme.	The Battleship Extreme shall respond to the mouse commands given by the Player.		
TC_03	There should be a proper functioning of an intelligent AI. **PRECONDITON:** User is logged into the game**,** Game mode - Human Vs. Computer mode. Turn = AI/Computer.	The computer shall destroy/make a move on its turn against the Player.		
TC_04	There should be virtual currency for purchasing the weapons for inventories. **PRECONDITION:** TC02 is complete and Player plans to buy weapons.	Virtual currency counter shall be displayed on the play screen and shall be updated when Player buys any weapon.		
TC_05	There should be a cost associated with every weapon and the cost of each weapon should be determined by the amount of damage caused by that weapon. **PRECONDITION:** TC_04 is complete and Player presses "A".	Attack Mechanism Select screen shall be displayed and there shall be a cost associated with every weapon.		

TC_06	The virtual currency should be used to move the Player's own ship. **PRECONDITION:** TC_05 is complete and Player selects "MoveShip" as a weapon, PlayerCurrencyCounter > Cost of "MoveShip"	"MoveShip" weapon shall be selected, the currency counter shall be updated and Player shall be able to move his/her own FleetShip.		
TC_07	The virtual currency should be used to skip the Opponent Player's turn. **PRECONDITION**: TC_05 is complete and Player selects "SkipTurn" as a weapon.PlayerCurrencyCounter > Cost of "SkipTurn"	"SkipTurn" weapon shall be selected, the currency counter shall be updated and Player shall be able to take his/her turn again.		
TC_08	The Ore refineries should generate money at the end of every move. **PRECONDITION:** User is logged into the game. Turn = Human/Player. Mode = Human vs. AI. Player makes a HIT.	The virtual currency counter shall be increased by 20 units.		
TC_09	There should be a screen for notifications which updates the Player about the moves made by the Opponent Player. **PRECONDITION**: User is logged into the game. Turn = AI. Mode = Human vs. AI. AI purchases a weapon.	The notification screen of the Player shall be updated.		
TC_10	Test to check the working of Attack Screen Pop-up. **PRECONDITION:** User is logged into the game. Turn = Player. Mode = Human vs. AI. Player presses "A".	A selectWeapon screen to select the attack mechanism should pop up.		
TC_11	Test to check the number of Attack Mechanisms. **PRECONDITION:** TC_10 is complete.	There shall be four Attack mechanisms available to the Player : Giga Bombs, Mega Bombs, Homing Missiles, and Radars		

TC_12	Test to check cost of Attack Weapons. **PRECONDITION:** TC_10 is complete.	The cost of weapons shall be in the following order: Giga Bombs > Mega Bombs > Homing Missile > Radar		
TC_13	Test to check GigaBomb functioning. **PRECONDITION:** TC_10 is complete and selectedWeapon = "GigaBomb".	Player shall be able to select a cell in the Opponent's territory using mouse and then press "H" to destroy Opponent's territory		
TC_14	Test to check GigaBomb functioning. **PRECONDITION:** TC_13 is complete, attack = HIT.	The whole ship shall sink and the Player shall be notified of the change via notification screen.		
TC_15	Test to check MegaBomb functioning. **PRECONDITION**: TC_10 is complete and selectedWeapon = "MegaBomb".	Player shall be able to select a grid of 3X3 cell in the Opponent's territory using mouse and then press "H" to destroy Opponent's territory.		
TC_16	Test to check MegaBomb functioning. **PRECONDITION:** TC_15 is complete, attack = HIT.	The whole ship shall sink if any ship is within the range and the Player shall be notified of the change via notification screen.		
TC_17	Test to check HomingMissile functioning. **PRECONDITION:** TC_10 is complete and selectedWeapon = "HomingMissile".	Player shall be able to select a grid of 3X3 cell in the Opponent's territory using mouse and then press "H" to destroy Opponent's territory.		
TC_18	Test to check HomingMissile functioning. **PRECONDITION:** TC_17 is complete, attack = HIT.	One section of a ship (at random) shall be hit if it is in range and the Player shall be notified if it a HIT or a full MISS via notification screen.		
TC_19	Test to check Radar functioning. **PRECONDITION**: TC_10 is complete and selectedWeapon = "Radar".	Player shall be able to select a grid of 3X3 cell in the Opponent's territory using mouse and then press "H" to probe the Opponent's territory.		
TC_20	Test to check Radar functioning. **PRECONDITION:** TC_19 is complete, attack = HIT.	If any ship is present in that area then the Player shall be notified.		

TC_21	Test to check the working of Defense Screen Pop-up. **PRECONDITION:** User is logged into the game. Turn = Player. Mode = Human vs. AI. Player presses "D".	A selectWeapon screen to select the defense mechanism should pop up.		
TC_22	Test to check the number of Defense Mechanisms. **PRECONDITION:** TC_21 is complete.	There shall be three defense mechanisms available to the Player: MoveShip, SkipTurn and Shield.		
TC_23	Test to check MoveShip functioning. **PRECONDITION:** TC_22 is complete.selectedWeapon = "MoveShip".	The Player shall be able to move their ships by maximum of 5 cells to the left, right, up, or down, and the board shall be updated.		
TC_24	Test to check SkipTurn functioning. **PRECONDITION**: TC_22 is complete.selectedWeapon = "SkipTurn".	The Player shall be able to play his/her turn again and opponent Player should not get his/her turn.		
TC_25	Test to check Shield functioning. **PRECONDITION:** TC_22 is complete.selectedWeapon = "Shield".	The Opposite Player shall not be able to do any destruction to the Player's Ships if Opposite Player buys any attack mechanism.		

Test Cases for Non-functional Requirements

ID	TEST DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	RESULT
TC_26	Check for Speed, Latency, Accuracy and Capacity. **Precondition:** The user is logged into the social networking account. User has opted to play Battleship Extreme. Mode = Player vs. Computer. User has entered his/her info. User selects "Play Game" on main screen.	The game should be loaded within 10 seconds.		
TC_27	Check for Speed, Latency, Accuracy and Capacity. **Precondition:** TC_26 is complete. User presses "A" to buy weapons.	The Attack Weapons Mechanism screen should be loaded within 3 seconds.		

TC_28	Check for Speed, Latency, Accuracy and Capacity. **Precondition:** TC_27 is complete.weaponSelected = "GigaBomb" and Player presses "H"	The response time of the command for the Player's execution shall be less than 150 milliseconds.		
TC_29	Check for Speed, Latency, Accuracy and Capacity. **Precondition:** TC_28 is complete.	The notifications in notification table shall be updated within 200 milliseconds after the opponent makes a move.		
TC_30	Check for Speed, Latency, Accuracy and Capacity. **Precondition:** The game is in test mode and a tester is testing the game using traffic simulation software.	The game should support at least 1000 simultaneous Players playing concurrently at a point of time.		
TC_31	**Load Testing**: This tests reliability under unexpected or rare workloads	The game shall display a menu offering the player to go to the game controls page in case the player enters an invalid input and repeat this process until the player enters a valid move.		
TC_32		The game should not go haywire in case the player enters invalid inputs and instead offer help to the player.		
TC_33		The game should continue to operate in the local mode for 5 minutes whenever it loses its link to the central server or the social networking site on which it is hosted goes down.		
TC_34	**Portability Testing**: This tests the Software with intent that it should be re-useable and can be moved to another configuration as well.	The Battleship Extreme must run on all the widely used operating systems as long as the system has a java enabled web browser.		
TC_35		The game must support modern operating systems like Windows 7/8/XP or Unix-like OSs (Ubuntu, Solaris, Fedora, Debian etc.) or MacOS.		
TC_36		The game must be able to run in an incessant manner with web browsers like Internet Explorer, Mozilla Firefox, Google Chrome and Apple Safari.		

TC_37	**Security Testing:** This tests that the system protects its data and maintains functionality as intended.	The functionality and data should be accessible only by the development and test manager.		
TC_38		The managers should have access to data such as data about high scores, data displayed on grids, notification logs, data about defense weapons inventory etc. in order to ensure the correct functioning of the game.		
TC_39		The development and test manager should not have access to sensitive data like the information entered by the player while creating an account.		
TC_40		The players' personal information should be accessible only to the business development manager.		
TC_41		The data should be accessible only by the database administrator		
TC_42		There should be clustered file-systems in order to ensure that any data should not be lost.		
TC_43		There should be periodic check every 24 hours in order to ensure the integrity of data.		
TC_44		The game should protect private information in accordance with the relevant privacy laws and the company's information policy.		
TC_45		No player should be allowed give command or control for any other player.		
TC_46		There should be a periodic scanning (once in every 12 hours) on the system on which game is hosted in order to ensure protection against any malicious interference like virus, worms, and Trojan horses, among others.		

TC_47	**Usability Testing**: This Tests and verifies the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.	The user should be able to learn all the controls within 6 minutes after registration.		
TC_48		The user should not change the controls for playing the game.		
TC_49		The user should have options to stop or mute the background music.		
TC_50		There should be a hyperlink to the instructions on how to play the game on the main menu.		
TC_51	**GUI Software Testing:** This tests the game to ensure it meets its written specifications for GUI.	The user should feel the emulation of the actual battlefield by using dark colors and images of war-like objects like ships, missiles, bombs, radars etc.		
TC_52		The game should have music rich with melodies and feelings and should support a war-like theme.		
TC_53		The user should also feel that he/she is playing the game on board.		

38 Testing Schedule

Type of Testing	Start Date	End Date	Comments
Unit testing	1/13/2014	2/14/2014	End of Release 1. Unit Testing shall be done by developers parallel to the development of numerous components.
Unit testing	2/21/2014	3/25/2014	End of Release 2.
Integration Testing	3/26/2014	4/13/2014	The entire individually developed component should be integrated and tested.
Regression Testing	4/14/2014	4/20/2014	After all the components are integrated, Regression Testing should be done in order to ensure that any new defects after integration have not been introduced.
System Testing	4/21/2014	4/24/2014	The system testing is to be performed by team members to ensure that the system has turned out how it was supposed to.
User Acceptance Testing	4/25/2014	4/27/2014	User acceptance testing is performed by the customer to make sure the system does what they think it is supposed to do/expect it to do.

39 References/Bibliography

- https://www.draw.io/
- www.creately.com
- http://www.uml-diagrams.org/package-diagrams-examples.html
- Bernd Bruegge, Allen H. Dutoit Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd Edition
- http://www.cs.uic.edu/~i440/
- http://msdn.microsoft.com/en-us/library/ee658124.aspx