## **VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY INTERNATIONAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**



**DATA STRUCTURES AND ALGORITHMS**

Instructor: **MSc. Nguyen Trung Nghia**

Lecturer: **Dr. Tran Thanh Tung**

**PROJECT: BATTLESHIP GAME**

**GROUP NAME: PANADOL EXTRA**

|  |  |
| --- | --- |
| **Student Name** | **Student ID** |
| Huynh Ngoc Anh Thu | ITCSIU21034 |
| Nguyen Tan Phat | ITITIU21354 |
| Pham Nguyen Dang Khoi | ITCSIU21196 |

# **TABLE OF CONTENT**

[**VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY INTERNATIONAL UNIVERSITY** 1](#_Toc156067599)

[**TABLE OF CONTENT** 2](#_Toc156067600)

[**LIST OF TABLES** 6](#_Toc156067601)

[**LIST OF FIGURES** 7](#_Toc156067602)

[**EXECUTIVE SUMMARY** 9](#_Toc156067603)

[**1.** **Project Overview:** 9](#_Toc156067604)

[**2.** **Objectives and Scopes:** 9](#_Toc156067605)

[***2.1 Objectives:*** 9](#_Toc156067606)

[***2.2 Scope:*** 9](#_Toc156067607)

[**3.** **Final Outcomes:** 10](#_Toc156067608)

[**CHAPTER I: INTRODUCTION** 11](#_Toc156067609)

[**1.** **Background and Context:** 11](#_Toc156067610)

[**2.** **Goals and Objectives:** 11](#_Toc156067611)

[**CHAPTER II: PROJECT PLANNING** 12](#_Toc156067612)

[**1.** **Timeline and Milestones:** 12](#_Toc156067613)

[12](#_Toc156067614)

[***1.1 Ideas Collecting Process:*** 12](#_Toc156067615)

[***1.2 Coding Process:*** 13](#_Toc156067616)

[***1.3 Bug-Fixing and Testing Process:*** 13](#_Toc156067617)

[***1.4 Slide Designing Process:*** 13](#_Toc156067618)

[***1.5 Report Writing Process:*** 14](#_Toc156067619)

[**2.** **Members’ Roles and Responsibilities:** 14](#_Toc156067620)

[**3.** **Risk Assessment and Mitigation:** 14](#_Toc156067621)

[***3.1 Technical Challenges:*** 14](#_Toc156067622)

[***3.2 Lack of Group Member Responsibility:*** 15](#_Toc156067623)

[***3.3 Resource Constraints:*** 15](#_Toc156067624)

[***3.4 User Acceptance:*** 15](#_Toc156067625)

[***3.5 Integration Challenges:*** 15](#_Toc156067626)

[**CHAPTER III: DESIGN AND ARCHITECTURE** 16](#_Toc156067627)

[**1.** **System’s Architecture Overview:** 16](#_Toc156067628)

[***1.1 Game Logics Overview:*** 16](#_Toc156067629)

[***1.2 GUI Overview:*** 16](#_Toc156067630)

[***1.3 Code Explanation: (https://github.com/panadolextra91/DSA-Project)*** 16](#_Toc156067631)

[*1.3.1 Ship:* 16](#_Toc156067632)

[*1.3.2 Treasure:* 17](#_Toc156067633)

[*1.3.3 Grids:* 19](#_Toc156067634)

[*1.3.4 Artificial Intelligence (AI):* 21](#_Toc156067635)

[*1.3.5 BackgroundPanel class* 27](#_Toc156067636)

[*1.3.6 Game Class* 29](#_Toc156067637)

[*1.3.7 GamePanel Class* 31](#_Toc156067638)

[*1.3.8 Graphics-ralated methods*: 35](#_Toc156067639)

[*1.3.9 Sound Effects:* 38](#_Toc156067640)

[*1.3.10 Videos:* 40](#_Toc156067641)

[**2.** **Diagrams:** 41](#_Toc156067642)

[***2.1 Behavioural Diagrams:*** 41](#_Toc156067643)

[*2.1.1 Use Case Diagram:* 41](#_Toc156067644)

[*2.1.1 Sequence Diagram:* 42](#_Toc156067645)

[***2.2 Structural Diagrams:*** 44](#_Toc156067646)

[*2.2.1 UML Diagram (Class Diagram):* 44](#_Toc156067647)

[*2.2.2 Entity Relationship Diagram (ERD):* 45](#_Toc156067648)

[***2.3 Swimlane Diagram:*** 47](#_Toc156067649)

[***2.4 System Diagrams:*** 49](#_Toc156067650)

[*2.4.1 Technology Diagram*: 49](#_Toc156067651)

[*2.4.2 Architecture Diagram:* 50](#_Toc156067652)

[**3.** **Key Designs:** 51](#_Toc156067653)

[***3.1 Entities:*** 51](#_Toc156067654)

[*3.1.1 Grids:* 51](#_Toc156067655)

[*3.1.2 Ships:* 52](#_Toc156067656)

[*3.1.3 Treasures:* 53](#_Toc156067657)

[*3.1.4 Game Panels:* 54](#_Toc156067658)

[***3.2 Levels:*** 57](#_Toc156067659)

[*3.2.1 Easy:* 57](#_Toc156067660)

[*3.2.2 Medium:* 57](#_Toc156067661)

[*3.2.3 Difficulty:* 58](#_Toc156067662)

[**CHAPTER IV: DEVELOPMENT** 59](#_Toc156067663)

[**1.** **Development Process’s Description:** 59](#_Toc156067664)

[***1.1 Ideas Collecting:*** 59](#_Toc156067665)

[***1.2 Code Process:*** 59](#_Toc156067666)

[***1.3 Bug Fixing – Testing:*** 59](#_Toc156067667)

[**2.** **Programming languages, Frameworks and Tools:** 59](#_Toc156067668)

[***2.1. IDE:*** 59](#_Toc156067669)

[***2.2. Language:*** 59](#_Toc156067670)

[***2.3. Support:*** 59](#_Toc156067671)

[**3.** **Challenges:** 60](#_Toc156067672)

[**4.** **Code Management and Version Control:** 60](#_Toc156067673)

[***4.1 Version Control with Git:*** 60](#_Toc156067674)

[***4.2 Collaborative Workflow:*** 60](#_Toc156067675)

[***4.3 Contributor Management:*** 60](#_Toc156067676)

[***4.4 Group Chat on Messenger:*** 60](#_Toc156067677)

[**CHAPTER V: TESTING AND BUG TRACKING** 61](#_Toc156067678)

[**1.** **Testing Methodologies:** 61](#_Toc156067679)

[**2.** **Results and Bug Tracking:** 61](#_Toc156067680)

[**3.** **Quality Assurance Measures:** 62](#_Toc156067681)

[***3.1 Bug-Tracking Process:*** 62](#_Toc156067682)

[*3.1.1 Bug Identification:* 62](#_Toc156067683)

[*3.1.2. Regression Testing:* 62](#_Toc156067684)

[***3.2 Quality Assurance Measures:*** 62](#_Toc156067685)

[*3.2.1. Test Coverage:* 62](#_Toc156067686)

[*3.2.2. Performance Metrics:* 62](#_Toc156067687)

[*3.2.3. Code Reviews:* 62](#_Toc156067688)

[*3.2.4. Compliance Checks:* 62](#_Toc156067689)

[**CHAPTER VI: GAME INSTRUCTION** 63](#_Toc156067690)

[**1.** **Begin the Game:** 63](#_Toc156067691)

[**2.** **Let's play!** 63](#_Toc156067692)

[**3.** **Victory or Defeated?** 63](#_Toc156067693)

[**CHAPTER VII: LESSONS LEARNED** 64](#_Toc156067694)

[**1.** **Lesson Learned: Project Monitoring and Task Management** 64](#_Toc156067695)

[**2.** **Navigating Challenges: Dealing with Team Dynamics** 64](#_Toc156067696)

[**3.** **Balancing Independence and Collaboration: Individual Contributions with Collective Responsibility** 64](#_Toc156067697)

[**CHAPTER VIII: CONCLUSION** 65](#_Toc156067698)

[**1.** **Summary of the Outcomes:** 65](#_Toc156067699)

[***1.1 Eye-Catching GUI:*** 65](#_Toc156067700)

[***1.2 Smoothly Run Game Logics:*** 65](#_Toc156067701)

[***1.3 High-Quality Images:*** 65](#_Toc156067702)

[***1.4 Enjoyable Sounds and Videos:*** 65](#_Toc156067703)

[***1.5 Easy to Maintain Source Code:*** 65](#_Toc156067704)

[**2.** **Closing Remarks:** 65](#_Toc156067705)

[**CHAPTER IX: APPENDICES** 67](#_Toc156067706)

[**1.** **Glossary of Terms:** 67](#_Toc156067707)

[**2.** **GitHub Related Insights:** 67](#_Toc156067708)

[68](#_Toc156067709)

[**REFERENCES** 69](#_Toc156067710)

# **LIST OF TABLES**

[Table 1: Role and Responsibilities 14](#_Toc156067711)

[Table 2: Test Case 1 61](#_Toc156067712)

[Table 3: Test Case 2 61](#_Toc156067713)

[Table 4: Test Case 3 61](#_Toc156067714)

[Table 5: Test Case 4 61](#_Toc156067715)

[Table 6: Test Case 5 62](#_Toc156067716)

[Table 7: Test Case 6 62](#_Toc156067717)

[Table 8: Glossary of Terms 67](#_Toc156067718)

# **LIST OF FIGURES**

[Figure 1: Gantt Chart 12](#_Toc156067719)

[Figure 2: Segments 16](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067720)

[Figure 3: BOAT\_SIZES 16](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067721)

[Figure 4: Ship Constructor 17](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067722)

[Figure 5: toggleSideways method 17](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067723)

[Figure 6: isDestroyed method 17](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067724)

[Figure 7: List of Treasures 18](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067725)

[Figure 8: initializeTreasures method 18](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067726)

[Figure 9: isTreasureAtPosition method 18](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067727)

[Figure 10: markTreasureAsOpened method 19](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067728)

[Figure 11: CELL\_SIZE 19](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067729)

[Figure 12: Grid Dimension 19](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067730)

[Figure 13: Marker 19](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067731)

[Figure 14: SelectionGrid constructor 19](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067732)

[Figure 15: placeShip method 20](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067733)

[Figure 16: markPosition method 20](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067734)

[Figure 17: allShipDestroyed method 21](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067735)

[Figure 18: SimpleRandomAI constructor 21](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067736)

[Figure 19: SmarterAI constructor 21](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067737)

[Figure 20: selectMove method 22](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067738)

[Figure 21: selectMove method of SmarterAI 23](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067739)

[Figure 22: getSmartAttack method 24](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067740)

[Figure 23: getSmartAttack method 24](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067741)

[Figure 24: findMostOpenPosition method 24](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067742)

[Figure 25: getAdjacentNotAttackCount method 24](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067743)

[Figure 26: atLeastTwoHitsInDirection method 25](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067744)

[Figure 27: getAdjacentSmartMoves method 25](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067745)

[Figure 28: getAdjacentCells method 25](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067746)

[Figure 29: updateShipHits method 26](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067747)

[Figure 30: containsAllPositions method 26](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067748)

[Figure 31: reset method 26](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067749)

[Figure 32: reset method of SmarterAI 27](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067750)

[Figure 33: drawTextOnPanel method 27](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067751)

[Figure 34: paintComponent method 28](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067752)

[Figure 35: startTextAnimation method 28](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067753)

[Figure 36: splitText method 29](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067754)

[Figure 37: Game class Key Components 29](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067755)

[Figure 38: Game constructor 30](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067756)

[Figure 39: startGame method 30](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067757)

[Figure 40: GameState Enum 31](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067758)

[Figure 41:: SelectionGrid for Player and Computer 32](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067759)

[Figure 42: aiController 32](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067760)

[Figure 43: statusPanel 32](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067761)

[Figure 44: GamePanel constructor 32](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067762)

[Figure 45: GamePanel (2) 33](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067763)

[Figure 46: GamePanel (1) 33](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067764)

[Figure 47: paint method 33](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067765)

[Figure 48: handleInput method 34](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067766)

[Figure 49: mouseReleased method 34](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067767)

[Figure 50: mouseMoved method 34](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067768)

[Figure 51: drawRadarBackground method 35](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067769)

[Figure 52: paint method in GamePanel class 35](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067770)

[Figure 53: paint method in Marker class 36](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067771)

[Figure 54: drawGrid method (2) 37](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067772)

[Figure 55: drawGrid method (1) 37](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067773)

[Figure 56: drawTreasures method 38](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067774)

[Figure 57: playSound method 39](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067775)

[Figure 58: Prepare the audio files 39](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067776)

[Figure 59: playVideo method (2) 40](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067777)

[Figure 60: playVideo method (1) 40](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067778)

[Figure 61: Use Case Diagram 41](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067779)

[Figure 62: Sequence Diagram 42](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067780)

[Figure 63: UML Diagram 44](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067781)

[Figure 64: ERD Diagram 46](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067782)

[Figure 65: Swimlane Diagram 48](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067783)

[Figure 66: Technology Diagram 49](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067784)

[Figure 67: Architecture Diagram 50](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067785)

[Figure 68: Game Window with Grids 51](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067786)

[Figure 69: Computer Grid with Ships and Treasures in Debugmode 52](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067787)

[Figure 70: Ships vertically 53](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067788)

[Figure 71: Ships horizontally 53](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067789)

[Figure 72: Marker on Hit, Miss, Treasure and Sunk Ship 53](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067790)

[Figure 73: Closed Treasure 54](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067791)

[Figure 74: Opened Treasure 54](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067792)

[Figure 75: Menu Panel 55](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067793)

[Figure 76: Strategy Panel 56](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067794)

[Figure 77: Difficulty Panel 57](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067795)

[Figure 78: Contribution to branch main (commits) 67](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067796)

[Figure 79: Code Frequency 68](file:////Users/huynhngocanhthu/Desktop/DSA%20report%20-%20Panadol%20Extra%20Group.docx#_Toc156067797)

# **EXECUTIVE SUMMARY**

## **Project Overview:**

The project undertakes the digitization of the conventional Battleship game, presenting an immersive and interactive digital rendition. The singularly focused gameplay is extended into a single-player mode with three distinct difficulty levels, thereby providing users with a nuanced and challenging gaming experience. A meticulously designed scoring system has been integrated to quantify and exhibit the player's performance, contributing to an analytical aspect of the gaming environment.

User experience is a pivotal consideration, and as such, the project prioritizes intuitive and user-friendly interfaces. These interfaces are crafted to ensure seamless interaction and engagement, aligning with contemporary design principles. Furthermore, the incorporation of audio-visual elements, including humorous sound effects and videos upon game conclusion, adds an entertaining dimension to the overall user experience.

From a technical standpoint, the project leverages Java programming for the implementation of intricate game logic, while the Swing framework is employed to construct graphical interfaces. Multimedia components, encompassing sound effects and videos, contribute to a dynamic and multi-sensory gaming atmosphere.

## **Objectives and Scopes:**

### ***2.1 Objectives:***

* **Digital Transformation:** Transform the traditional Battleship game into a digital platform, leveraging Java and Swing for enhanced user interaction and a modern gaming experience.
* **Single-Player Variety:** Introduce three distinct difficulty levels in single-player mode to cater to players with varying skill levels, providing a challenging and engaging experience.
* **Scoring System Implementation:** Develop and implement a scoring system to evaluate player performance, adding a competitive element and encouraging strategic gameplay.
* **User-Friendly Interfaces:** Design intuitive and user-friendly interfaces to ensure accessibility and ease of navigation, enhancing the overall gaming experience for players of all backgrounds.
* **Multimedia Integration:** Enhance player engagement by incorporating entertaining sound effects and videos that dynamically respond to game outcomes, adding a layer of enjoyment to victories and losses.

### ***2.2 Scope:***

* **Game Development:** Focus on implementing the core mechanics of the Battleship game in a digital environment, ensuring gameplay authenticity and adherence to the classic rules.
* **Difficulty Levels:** Define and implement three difficulty levels for the single-player mode, offering a range of challenges to accommodate players with varying levels of expertise.
* **Scoring System Design:** Create a scoring system that factors in elements such as accuracy, speed, and strategic decisions, providing players with a quantifiable measure of their performance.
* **Interface Design:** Develop user-friendly interfaces with clear layouts and navigation, making the game accessible to a broad audience and ensuring a positive user experience.
* **Multimedia Integration:** Integrate sound effects and videos that complement game events, contributing to a more immersive and enjoyable gaming atmosphere.
* **Platform Compatibility:** Ensure the project's compatibility with different platforms, providing flexibility for users to enjoy the game on various devices.
* **Testing and Debugging:** Conduct thorough testing to identify and rectify potential bugs or issues, ensuring a polished and seamless gaming experience for end-users.
* **Documentation:** Create comprehensive documentation that outlines the project's functionalities, design choices, and implementation details, facilitating future maintenance and potential expansion.

## **Final Outcomes:**

The culmination of this project is a seamlessly executed digital platform game developed using the Java programming language. The game boasts an engaging and immersive experience, enhanced by multimedia players that bring sound effects and videos to dynamically respond to game events. The graphical user interface (GUI) stands out with its attractive design, featuring excellent image resolution that contributes to the overall visual appeal. Each difficulty level in single-player mode is powered by distinct artificial intelligence (AI) algorithms, providing diverse and challenging experiences for players. The game's underlying logics run smoothly, ensuring a flawless gaming experience free from bugs or errors, a testament to the meticulous development and testing processes employed throughout the project. The result is a polished and enjoyable digital adaptation of the classic Battleship game, delivering entertainment to a broad audience across various platform.

# **CHAPTER I: INTRODUCTION**

## **Background and Context:**

The background of this project lies in the pursuit of transforming the timeless Battleship game into a modern digital platform, seamlessly blending classic strategy with cutting-edge technology. Rooted in game development, the endeavour is driven by the ambition to provide players with an immersive and dynamic experience. Leveraging Java programming, the project introduces multimedia elements, including captivating sound effects and videos, to elevate the visual and auditory dimensions of gameplay. The decision to incorporate multiple difficulty levels, each supported by distinct AI algorithms, reflects a commitment to catering to a diverse player audience. Positioned within the broader landscape of gaming industry trends, this initiative not only pays homage to the nostalgia of traditional board games but also aligns with the contemporary shift towards revitalizing classic titles for a new era of gamers. In essence, the project aspires to create an updated Battleship experience that remains faithful to its strategic origins while embracing the innovation and expectations of today's digital gaming landscape.

## **Goals and Objectives:**

* **Digital Transformation:** Transform the traditional Battleship game into a digital platform, providing players with a contemporary and engaging gaming experience.
* **Multi-Level Single-Player Mode:** Implement three distinct difficulty levels in single-player mode, offering varying degrees of challenge to cater to different player skill levels.
* **Scoring System:** Introduce a comprehensive scoring system to track and reward player performance, enhancing the competitive aspect of the game.
* **User-Friendly Interfaces:** Design intuitive and user-friendly interfaces to ensure a seamless and enjoyable gaming experience for players of all ages.
* **Multimedia Integration:** Incorporate multimedia elements, including attractive GUI, high-resolution images, and entertaining sound effects/videos, to enhance the overall sensory appeal of the game.
* **Bug-Free Logic:** Develop a robust game logic that runs smoothly without bugs or errors, ensuring a stable and reliable gaming environment.
* **AI Algorithms:** Implement different AI algorithms for each difficulty level, providing intelligent and adaptive opponents for players to compete against.
* **Win/Loss Feedback:** Include dynamic sound effects and videos to provide engaging feedback to players upon winning or losing a game, enhancing the overall gaming atmosphere.
* **Innovation and Nostalgia:** Blend innovation with a sense of nostalgia, aligning the project with both modern gaming expectations and the classic charm of traditional board games.
* **Versatile Player Experience:** Strive to create a versatile player experience that caters to a broad audience, from casual gamers to those seeking more challenging strategic gameplay.

# **CHAPTER II: PROJECT PLANNING**

## **Timeline and Milestones:**

### 

Figure 1: Gantt Chart

### ***1.1 Ideas Collecting Process:***

* **Duration:** 09/12/2023 - 26/12/2023.
* **Description:** In the initial phase of our academic report, we engaged in a collaborative process where team members contributed diverse insights. We rigorously tested these ideas for alignment with our framework and project goals. Subsequently, we made informed decisions on which ideas to accept, considering factors like feasibility and relevance. The final step involved allocating tasks strategically, ensuring a streamlined approach towards realizing the chosen ideas.
* **Milestone:** Finalise the core entities and objects of the project.

### ***1.2 Coding Process:***

* **Duration:** 27/12/2023 - 07/01/2024
* **Description:** During the Coding Process, team members embarked on individual tasks using their preferred Integrated Development Environments (IDEs). This stage saw the transformation of conceptualized ideas into tangible code, with each member contributing to the collective effort. Upon completion of their tasks, team members pushed their commits to the GitHub repository dedicated to the project. The collaborative nature of this platform allowed for seamless integration of individual contributions. However, before finalizing the merge request, the project manager played a crucial role in the quality control process. The project manager meticulously reviewed each commit, checking for adherence to coding standards, overall project coherence, and compatibility with the established framework. Approval from the project manager served as a gatekeeper, ensuring that only validated and high-quality code was merged into the current branch. This thorough validation process not only maintained code integrity but also upheld the project's overall efficiency and reliability.
* **Milestone:** The project code is finished.

### ***1.3 Bug-Fixing and Testing Process:***

* **Duration:** 05/01/2024 - 11/01/2024
* **Description:** In the Bug-Fixing and Testing Process, our team actively engaged in identifying and resolving errors encountered during the coding and testing phases. Members played a crucial role in this collaborative effort, sharing insights about bugs they encountered. When a team member identified a bug and possessed a solution, they promptly communicated it through commit comments in the GitHub repository. This streamlined approach facilitated a quick resolution, promoting transparency and knowledge sharing within the team. However, when a code owner faced challenges without an immediate solution, a collaborative approach took centre stage. The code owner reached out to the project manager and fellow developers for assistance. This collaborative problem-solving dynamic ensured that challenges were addressed collectively, harnessing the diverse expertise within the team. Through effective communication and shared problem-solving, the Bug-Fixing and Testing Process not only rectified issues promptly but also fostered a collaborative team spirit, enhancing the overall quality and reliability of our project.
* **Milestone:** Fixed all the bugs and errors the developers encountered.

### ***1.4 Slide Designing Process:***

* **Duration:** 29/12/2023 - 10/01/2024
* **Description:** In the Slide Design process, the designated member selects a fitting template, incorporates project images and content, and shares the draft with the team. Each member provides input, and the slides undergo a crucial review by the project manager. This collaborative effort ensures a visually compelling and information-rich presentation aligned with our project objectives.
* **Milestone:** The slides are finished.

### ***1.5 Report Writing Process:***

* **Duration:** 11/01/2024 - 12/01/2024
* **Description:** In the Report Writing Process, our approach combines individual expertise and collective input to create a comprehensive document. The main writer initiates the process by selecting an appropriate template, adding essential content, and incorporating non-code-related images. For the technical sections, the development team takes the reins, with each member responsible for detailing the design and architecture of the specific parts they coded. This targeted approach ensures a deep dive into the intricacies of the project's development. The project manager contributes manager-related content, providing a holistic perspective on project management aspects. Subsequently, the entire team engages in a collaborative proofreading and feedback loop. This thorough review guarantees a polished and cohesive report that not only reflects the technical excellence of our project but also aligns seamlessly with managerial considerations.
* **Milestone:** The report is checked and finished.

## **Members’ Roles and Responsibilities:**

|  |  |
| --- | --- |
| **Student Name** | **Role** |
| Huynh Ngoc Anh Thu | Project Manager (PM)  Game Logics Developer  GUI Developer  Testing and Bug Fixing  Media Files Collector  Images Collector  Slide Maker  Diagrams and Charts  Report Writer  Presentation |
| Nguyen Tan Phat | Game Logics Developer  GUI Developer  Testing and Bug Fixing  Media Files Collector  Slide Maker  Report Writer  Presentation |
| Pham Nguyen Dang Khoi | Game Logics Design  GUI Developer  Testing and Bug Fixing  Images Collector  Report Writer  README Writer  Presentation |

Table 1: Role and Responsibilities

## **Risk Assessment and Mitigation:**

### ***3.1 Technical Challenges:***

* **Risk:** Unforeseen technical challenges may arise during the development phase, leading to delays.
* **Mitigation:** Conduct thorough research and feasibility studies before the project begins. Maintain a skilled and adaptable development team to address technical issues promptly.

### ***3.2 Lack of Group Member Responsibility:***

* **Risk:** Some team members may exhibit a lack of responsibility, leading to missed deadlines, incomplete tasks, and overall project delays.
* **Mitigation:**
* Clearly Define Roles and Responsibilities: Establish well-defined roles for each team member from the project's initiation. Clearly communicate expectations regarding individual responsibilities and contributions.
* Regular Monitoring and Reporting: Implement a system for regular check-ins and progress reports. Team leaders or project managers should consistently monitor each member's contributions to identify and address issues promptly.
* Foster Team Collaboration: Encourage a collaborative team environment where members support each other. Foster open communication to address concerns, distribute work effectively, and ensure that no team member feels overwhelmed.
* Set Milestones and Deadlines: Breakdown the project into milestones with associated deadlines. This will create a structured timeline and help tracking the progress.

### ***3.3 Resource Constraints:***

* **Risk:** Limited availability of resources, including time, personnel, or technology.
* **Mitigation:** Develop a realistic project timeline. Ensure that team members have the necessary skills and training. Consider outsourcing certain tasks if internal resources are insufficient.

### ***3.4 User Acceptance:***

* **Risk:** The game may not meet user expectations, leading to dissatisfaction.
* **Mitigation:** Involve end-users in the development process through feedback sessions and testing. Conduct beta testing to gather insights and make necessary adjustments based on user feedback.

### ***3.5 Integration Challenges:***

* **Risk:** Integration issues may arise when combining different multimedia elements and technologies.
* **Mitigation:** Conduct extensive testing of integrated components. Ensure compatibility and seamless interaction between multimedia players, GUI, and other elements.

# **CHAPTER III: DESIGN AND ARCHITECTURE**

## **System’s Architecture Overview:**

### ***1.1 Game Logics Overview:***

The entire codebase is constructed utilising the Java programming language regarding functionalities that are implemented for ship generation, grid generation, and various levels of artificial intelligence opponents. Furthermore, the project's data structure is developed using ArrayList, notably evident in the processes related to ship generation and target hits.

### ***1.2 GUI Overview:***

To deliver a well-designed gaming interface to players, the codebase was configured with JavaFX libraries, enabling the creation of video effects based on methods implemented in the Game Logic class, also referred to as GamePanel. Furthermore, the game underwent significant improvement with the addition of buttons, GIFs, and sound effects, marking a notable progression from its initial version.

### ***1.3 Code Explanation: (https://github.com/panadolextra91/DSA-Project)***

#### *1.3.1 Ship**:*

a. Ship Information:

The public attribute BOAT\_SIZES holds significant importance in shaping the Battleship game by defining the number of ships allocated to each participant and specifying the number of segments each ship occupies. This attribute serves as a crucial parameter, influencing the game's architecture, determining ship placement on the grid, and establishing regulations regarding the configuration of ships for each player. The values assigned to BOAT\_SIZES provide essential information, indicating that each player is assigned a fixed number of ships, specifically 5 in this case. Furthermore, the associated sizes (in segments or cells) for these ships are defined as 5, 4, 4, 3, and 2, respectively. This information is pivotal in maintaining balance and fairness in the game, contributing to a strategic and engaging gameplay experience.

Figure : Segments

Figure : BOAT\_SIZES

This private attribute, named segments, represents the number of segments in a ship, indicating how many cells it occupies on the game grid. In the context of the Battleship game, each ship is divided into segments, and the value of this attribute specifies the length or size of the ship. It is a fundamental parameter for determining the positioning, drawing, and behaviour of the ship within the game, influencing aspects such as collisions, destruction, and overall gameplay mechanics.

b. Constructor:

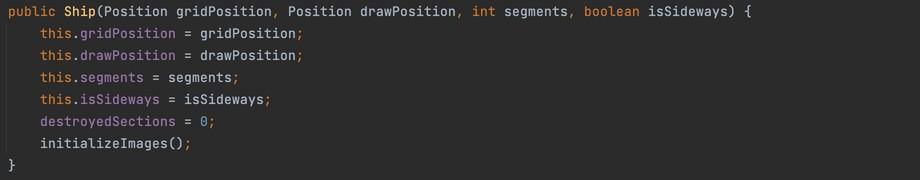
The constructor of the Ship class serves as the foundational element for creating instances of ships within the Battleship game. Designed to encapsulate essential ship attributes, this constructor initializes key properties, such as the ship's grid and draw positions, the number of segments it occupies, and its orientation (horizontal or vertical). Through this constructor, ships are instantiated with predefined characteristics, setting the stage for their integration into the game's grid system. The constructor's versatility allows for the dynamic creation of ships with varying attributes, contributing to the diversity and complexity of the game's naval fleet. It creates ships with default properties that are ready to use, assuming that the ship has already been placed when created.

Figure : Ship Constructor

* **Parameter gridPostion:** Encapsulates the position of a ship within the Battleship game grid, representing its location in terms of grid coordinates.
* **Parameter drawPosition:** Denotes the pixel coordinates of the ship's top-left corner, indicating the starting point for rendering within a cell.
* **Parameter segments:** Signifies the number of segments within a ship, effectively illustrating the extent of its coverage across grid cells.
* **Parameter isSideways:** Functions as a directional indicator, where a value of "true" signifies a horizontal orientation of the ship, while "false" denotes a vertical orientation. destroyedSections: The number of destroyed sections to help determine if all the ships have been destroyed when compared to segments.
* **InitializeImages method:** Serves a critical role in preparing the ship images for display based on the ship's characteristics. This method constructs the path for the ship image, considering the number of segments and orientation (horizontal or vertical). Subsequently, it reads the corresponding ship image from the file and applies a red colour filter, enhancing visual cues to signify ship destruction in the Battleship game.

c. toggleSideways method:

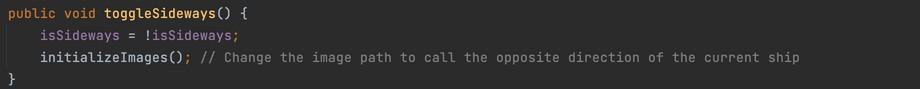
The toggleSideways method in the Ship class facilitates the dynamic alteration of a ship's orientation, allowing it to toggle between a horizontal and vertical state. This functionality is pivotal for strategic ship placement and provides players with flexibility in adapting their naval formations during the Battleship game.

Figure : toggleSideways method

d. isDestroyed method:

The isDestroyed method in the Ship class plays a pivotal role in evaluating whether all segments of the ship have been destroyed. By examining the number of sections destroyed in comparison to the total segments, this method provides a crucial check to determine if the ship is destroyed in the Battleship game. The return value is 'true' if all sections have indeed been destroyed, signalling the ship's elimination from the game.

Figure : isDestroyed method

#### *1.3.2 Treasure:*

The Treasure object within the Battleship game constitutes elements of strategic importance concealed within the game milieu. These entities serve as pivotal components, contributing to the nuanced dynamics of gameplay. This section delves into a scholarly examination of the inherent attributes and functionalities of the Treasure object, elucidating its role in augmenting the overarching strategic dimensions inherent to the gaming experience.

a. Treasure Information:

This part provides a succinct, yet comprehensive overview of key aspects related to treasures in the Battleship game. Delving into visual representations, dynamic states, and the numerical count of treasures, this section offers a concise exploration of these pivotal elements, enhancing readers' understanding of their roles within the gaming context.

* **Treasures Position:**

The private List<Position>treasures serves as a repository for storing the positions of treasures in the Battleship game. In this design, each treasure corresponds to a single cell, simplifying the implementation by avoiding the necessity for a separate class dedicated to treasures the fundamental principle that one treasure aligns with one cell and, consequently, one position, the list efficiently manages the spatial coordinates of treasures within the game.



Figure : List of Treasures

* **Treasures Initialisation:**

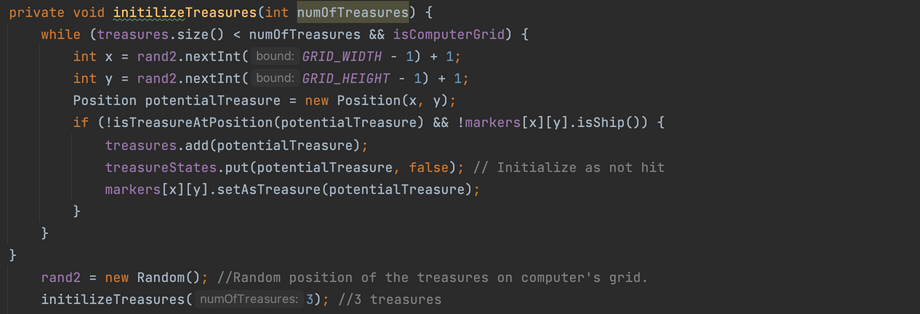
Method initializeTreasures randomly places a specified numOfTreasures on the grid, ensuring a non-overlapping distribution. The placement excludes the label column and row to maintain clarity and prevent interference with other game elements. This method contributes to the dynamic nature of the game, introducing an element of uncertainty and challenge for players as they navigate the grid in search of treasures. The initial number of treasures in each game is assigned to be 3.

Figure : initializeTreasures method

b. isTreasureAtPosition method:

A screen shot of a computer

Description automatically generatedThe method isTreasureAtPosition assesses whether a treasure occupies the designated position, providing a crucial function for players seeking to uncover hidden treasures during gameplay. The method will return 'true' if there is treasure at the checking position and will return 'false' otherwise.

Figure : isTreasureAtPosition method

* **Parameter pos:** The position to check for treasure.

c. markTreasureAsOpened method:

A black background with white text

Description automatically generatedRegisters the status of a treasure as 'opened' at the specified position on the game grid. The markTreasureAsOpened method plays a pivotal role in tracking the game's progress, allowing for the identification of explored treasures.

Figure : markTreasureAsOpened method

#### *1.3.3 Grids:*

a. Grid Information:

The Grid in the Battleship game is characterized by various properties that define its structure and appearance. These properties play a crucial role in managing the placement of ships, markers, and treasures on the grids.

* **Cell Size:**

The CELL\_SIZE constant determines the size of each cell in pixels, providing a basic unit for positioning ships, markers, and treasures within the grid.

Figure : CELL\_SIZE

* **Grid Dimension:**

Constants such as GRID\_WIDTH and GRID\_HEIGHT represent the number of cells horizontally and vertically, respectively, forming the overall dimensions of the game grid.



Figure : Grid Dimension

* **Marker System:**

The grid employs a marker system to track hits, misses, treasures, and other significant events during gameplay. The markers array is a fundamental component responsible for managing and visualizing these markers on the grid.

Figure : Marker

A screenshot of a computer program

Description automatically generatedb. Constructor:

Figure : SelectionGrid constructor

The constructor for the SelectionGrid class initializes a grid object with specified coordinates and additional properties, tailoring its behaviour based on whether it represents the player's or the computer's grid. This constructor serves as a crucial element in creating instances of the grid to be used in the Battleship game.

* **Parameter x:** The x-coordinate of the grid.
* **Parameter y:** The y-coordinate of the grid.
* **Parameter isComputerGrid:** Boolean indicating if the grid belongs to the computer. The value true of the grid is the computer's grid, and false is if the grid is the player's grid.

c. placeShip method:

The placeShip method in the SelectionGrid class facilitates the placement of ships on the game grid. The method allows for the strategic positioning of a ship object based on specified coordinates and orientation.



Figure : placeShip method

* **Parameter ship:** The Ship object to be placed.
* **Parameter gridX:** The x-coordinate in the grid.
* **Parameter gridY:** The y-coordinate in the grid.

**Add Ship to List (ships.add(ship)):**

* The Ship object passed as a parameter is added to the list of ships (List<Ship> ships), which keeps track of all the ships on the grid.

**Horizontal Ship Placement (if (ship.isSideways())):**

* If the ship is placed horizontally (sideways), the code enters the block.
* It iterates over each segment of the ship horizontally (for (int x = 0; x < ship.getSegments(); x++)).
* For each segment, it sets the corresponding cell in the markers grid as a ship marker using the ship reference (markers[gridX + x][gridY].setAsShip(ships.get(ships.size() - 1))).

**Vertical Ship Placement (else block):**

* If the ship is placed vertically, the code enters the block.
* It iterates over each segment of the ship vertically (for (int y = 0; y < ship.getSegments(); y++)).
* For each segment, it sets the corresponding cell in the markers grid as a ship marker using the ship reference (markers[gridX][gridY + y].setAsShip(ships.get(ships.size() - 1))).

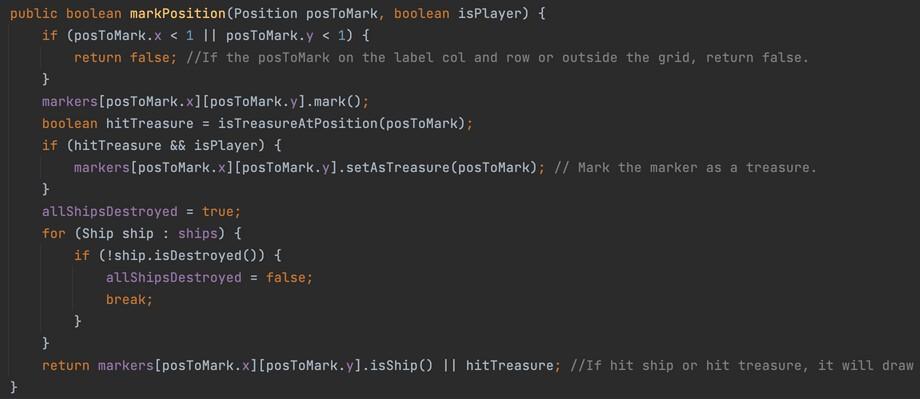
d. markPosition method:

Figure : markPosition method

The markPosition method in the SelectionGrid class plays a pivotal role in the Battleship game dynamics. This method allows for the marking of a specified grid position, indicating whether it hits a ship or uncovers a hidden treasure. Responsible for managing the game state, this method considers the validity of the specified position, updates the grid markers, and evaluates the impact on ships and treasures. Its versatile functionality extends to both player and computer interactions, contributing to the core gameplay experience by determining successful hits and the destruction of elements on the grid.

* **Parameter posToMark:** The position to be marked.
* **Parameter isPlayer:** Boolean indicating if the marking was made by the player.

**Check Position Validity (if (posToMark.x < 1 || posToMark.y < 1)):**

* Checks if the specified position is on the label column or row or outside the grid. If true, returns false.

**Mark the Position (markers[posToMark.x][posToMark.y].mark()):**

* Marks the specified position on the grid using the mark method of the corresponding.

**Marker object:**

* Check for Treasure Hit (boolean hitTreasure = isTreasureAtPosition(posToMark))
* Checks if the marked position hits a treasure by calling the isTreasureAtPosition method.

**Mark Treasure (if (hitTreasure && isPlayer)):**

* If the marked position hits a treasure and the marking is by the player, mark the marker as a treasure.

**Check All Ships Destroyed (for (Ship ship : ships)):**

* Iterates through all ships to check if any ship is not destroyed. Updates the allShipsDestroyed flag accordingly.
* Return Result (return markers[posToMark.x][posToMark.y].isShip() || hitTreasure)
* Returns true if the position hits a ship or treasure, false otherwise.

e. areAllShipsDestroyed method:

The method areAllShipDestroyed in the SelectionGrid serves as an indicator to check whether all the ships are destroyed. If yes, the method will return 'true', else, it will return 'false'.



Figure : allShipDestroyed method

#### *1.3.4 Artificial Intelligence (AI):*

In the realm of the Battleship game, the AI serves as the strategic mind controlling the computer's moves. This report delves into the implementation of two distinct AI classes, each offering a unique approach to gameplay. The first contender, SimpleRandomAI, embraces simplicity with randomized decision-making. On the other hand, SmarterAI represents a more sophisticated adversary, capable of adjusting its strategy based on the player's moves.

a. Constructor:



Figure : SimpleRandomAI constructor

* **Parameter playerGrid:** Reference to the player's grid to attack.

**Shuffle the list of valid moves randomly (Collections.shuffle(validMoves)):**

* A screen shot of a computer code

  Description automatically generatedThe validMoves list contains all possible positions on the grid, and shuffling it ensures that the AI's moves are selected in random order during gameplay.

Figure : SmarterAI constructor

* **Parameter playerGrid:** Reference to the player's grid to attack.
* **Parameter preferMovesFormingLine:** True will enable the smartest version of the AI to try and form lines when attacking ships.
* **Parameter maximiseAdjacentRandomisation:** True makes the randomised attacks prefer grid positions that have more not attacked points around them.

**Initializes the shipHits list (shipHits = new ArrayList<>();):**

* Keeps track of positions where ships were hit but not yet destroyed.

**Shuffle the list of valid moves randomly (Collections.shuffle(validMoves)):**

* The validMoves list contains all possible positions on the grid, and shuffling it ensures that the AI's moves are selected in a random order during gameplay.

b. selectMove methods:

Depending on which AI is called, the selectMove method will have some adjustments and advancements to enhance the game experience and challenge the players.

Figure : selectMove method

The method selectMove of the SimpleRandomAI will just take the top-most move of the list validMoves and then return it.

**Get the first position (Position nextMove = validMoves.get(0);):**

* This retrieves the first position from the list of valid moves (validMoves) and assigns it to the nextMove variable.

**Remove the selected move from the list of valid moves (validMoves.remove(0);):**

* Ensuring that the same position won't be selected again in subsequent calls.

**Return the selected move (return nextMove;):**

* The selected move (nextMove) is returned to be used in the gameplay.

The selectMove of the SmarterAI is an algorithm to select an appropriate move depending on whether any ships were currently hit and not yet destroyed. The AI will choose an attack adjacent to known ship-hit locations if a ship has been found, otherwise, it will select the next random move.A screenshot of a computer program

Description automatically generated

Figure : selectMove method of SmarterAI

**Position selectedMove;:**

* Declares a variable selectedMove to store the chosen move.

**Checks if there are ship hits(if (shipHits.size() > 0) { ... }):**

* If yes, it decides whether to choose a move based on forming a line (preferMovesFormingLine) or a smart attack (getSmartAttack).

**else { ... }:**

* If no ship is hit, it decides whether to maximize adjacent randomization (maximiseAdjacentRandomisation) or choose a random move.

**updateShipHits(selectedMove);:**

* Updates information about ship hits based on the selected move.

**validMoves.remove(selectedMove);:**

* Removes the selected move from the list of valid moves.

**return selectedMove;:**

* Finally, the selected move is returned.

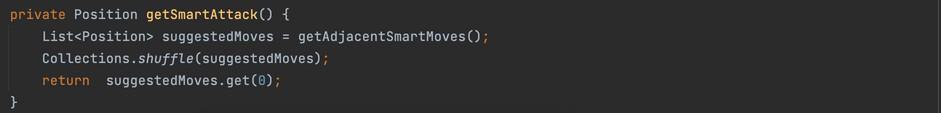


Figure : getSmartAttack method

A screen shot of a computer program

Description automatically generatedThe getSmartAttack method gets a list of moves adjacent to shipHits and chooses one at random.

Figure : getSmartAttack method

A screen shot of a computer code

Description automatically generatedThe getSmarterAttack method gets a list of moves adjacent to shipHits and chooses one based on whether it forms a line of at least two elements with adjacent ship hits. If no optimal guess is found a random adjacent move is selected. Finally, it will return a valid move that is adjacent to shipHits preferring one that forms a line.

Figure : findMostOpenPosition method

A computer screen with white text

Description automatically generatedThe method findMostOpenPosition searches for the valid move with the most adjacent cells that have not been attacked. Finally, it returns the first position with the highest score in the valid moves list.

Figure : getAdjacentNotAttackCount method

The method getAdjacentNotAttackCount counts the number of adjacent cells that have not been marked around the specified position.

* **Parameter position:** The position to count adjacent cells.

Finally, it will return the number of adjacent cells that have not been marked around the position.

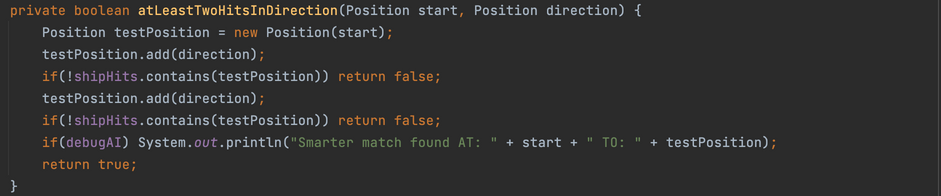
The method atLeastTwoHitsInDirection tests if there are two adjacent ship hits in a direction from a test start point.

Figure : atLeastTwoHitsInDirection method

* **Parameter start:** Position to start from (but not test).
* **Parameter direction:** Direction to move from the start position.

Finally, it will return true if there are two adjacent ship hits in the specified direction.

A screen shot of a computer screen

Description automatically generated

Figure : getAdjacentSmartMoves method

A screen shot of a computer program

Description automatically generatedThe method getAdjacentSmartMoves gets the adjacent cells around every shipHit and creates a unique list of the elements that are also still in the valid move list. Finally, it will return a list of all valid moves that are adjacent cells to the current ship hits.

Figure : getAdjacentCells method

The method getAdjacentCells creates a list of all adjacent cells around the position excluding any that are off the grid.

* **Parameter position:** Position to find adjacent cells around.

Finally, it will return a list of all adjacent positions that are inside the grid space.

A screen shot of a computer code

Description automatically generated

Figure : updateShipHits method

The method updateShipHits tests if the position hits a ship. Then evaluates if the ship that is hit would be destroyed. If it is destroyed the data is all cleared for that ship because it is no longer necessary to know about destroyed ships.

* **A screen shot of a computer

  Description automatically generatedParameter testPosition:** The position that is being evaluated for hitting a ship.

Figure : containsAllPositions method

The method containsAllPositions tests if all the positions in positionsToSearch are in

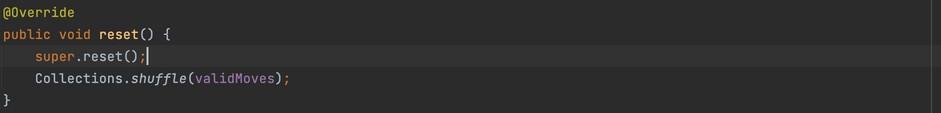
listToSearchIn.

* **Parameter positionsToSearch:** List of positions to search all.
* **Parameter listToSearchIn:** List of positions to search inside.

Finally, it will return true if all the positions in positionsToSearch are in listToSearchIn.

c. reset method:

Figure : reset method



The reset method of the SimpleRandomAI will reset the AI by resetting the parent class, and then reshuffling the refreshed list of valid moves.

A black rectangle with white text

Description automatically generated

Figure : reset method of SmarterAI

The reset method of the SmarterAI will reset the ships that have been hit and randomise the move order.

#### *1.3.5 BackgroundPanel class*

a. Purpose:

This class extends JPanel and is designed to display a background image with animated text overlay. It's typically used for creating visually appealing backgrounds in different parts of the game.

b. Key Components:

* **backgroundImage:** Holds the image that will be used as the panel's background.
* **VT323 Font:** A custom font used for drawing text.
* **textToDraw:** A list of strings that stores the text to be animated on the panel.
* **animationTimer:** A Timer that controls the animation of the text.

c. How it Works:

* When instantiated, the class loads an image and a custom font.

Figure : drawTextOnPanel method

* The drawTextOnPanel method allows specifying text and its maximum width, which is then split into lines and animated using the startTextAnimation method.A screen shot of a computer code

  Description automatically generatedThe paintComponent method is overridden to draw the background image and the animated text.

A computer screen shot of text

Description automatically generated

Figure : paintComponent method

d. BackgroundPanel Class Functions:

1. **Constructor BackgroundPanel(String imagePath):**

**Purpose:** Initializes the BackgroundPanel with a background image.

**Process:** Loads the background image from the given file path, sets up the custom font VT323, and initializes the animation timer.

**Key Operations:**

* Loading the background image using Toolkit.getDefaultToolkit().getImage(imagePath).Creating and registering the custom font.
* Setting up the Timer for text animation.

1. **startTextAnimation():**

**Purpose:** Begins the animation of text over the background.

**A screen shot of a computer program

Description automatically generatedProcess:** Resets the lines count and starts the animation timer, which triggers repaint() to update the panel.

Figure : startTextAnimation method

1. **drawTextOnPanel(String text, int maxWidth):**

**Purpose:** Prepares the text for animation.

**Process:** Splits the provided text into lines based on the maxWidth, then starts the text animation.

**Key Operation:** Utilizes the splitText method to divide the text into manageable lines.

A screen shot of a computer code

Description automatically generated

Figure : splitText method

1. **paintComponent(Graphics g):**

**Purpose:** Overrides the JPanel's paintComponent to draw the background image and animated text.

**Process:** Draws the background image and then iterates over the textToDraw list, drawing each line of text.

#### *1.3.6 Game Class*

a. Purpose:

This class serves as the main controller of the game. It initializes the main game window and handles the navigation between different game panels like the main menu, difficulty selection, and the game panel.

b. Key Components:

* **frame:** The main game window.
* **gamePanel:** An instance of GamePanel where the actual game takes place.
* **A black screen with green text

  Description automatically generatedGameDifficulty Enum:** Represents different difficulty levels for the game.

Figure : Game class Key Components

c. How it Works:

* The constructor sets up the main game window and initializes different panels.
* Methods like showStrategyPanel and showDifficultyPanel switch between different views in the game.
* startGame method initializes the game with a selected difficulty.
* Implements KeyListener to handle key events.

d. Game Class Functions

1. **Constructor - Game():**

**Purpose:** Sets up the main game window and panels.

**Process:** Initializes the JFrame, sets its properties, and adds the initial panel (BackgroundPanel).

**Key Operations:**

* Creating the main game frame.
* Adding a BackgroundPanel and button panel to the frame.
* A black screen with many colorful lines

  Description automatically generated with medium confidenceConfiguring the frame's properties (like size, visibility).

Figure : Game constructor

1. **startGame(GameDifficulty difficulty):**

**Purpose:** Starts a new game with the selected difficulty.

**Process:** Initializes GamePanel based on the chosen difficulty and updates the frame content to display the game.

**Key Operations:**

* Creating a new instance of GamePanel.
* Setting up key inputs for the game panel.
* A screen shot of a computer

  Description automatically generatedMaking the game panel visible.

Figure : startGame method

1. **showStrategyPanel() and showDifficultyPanel():**

**Purpose:** Switches to different panels for showing game strategy or selecting difficulty.

**Process:** These methods create and display panels for different game states, like showing the game strategy or difficulty options.

**Key Operations:**

* Creating new BackgroundPanel instances for different purposes.
* Setting up and displaying buttons for user interactions.

A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generated

Figure 41: showDifficultyPanel method

Figure 40: showStrategyPanel method

#### *1.3.7 GamePanel Class*

a. Purpose: This class is where the actual gameplay occurs. It manages the game's state, including ship placement, attacking, and tracking game progress.

b. Key Components:

* **A screen shot of a video game

  Description automatically generatedGameState Enum:** Represents different states of the game like placing ships, firing shots, and game over.

Figure : GameState Enum

* **computer and player:** Two SelectionGrid objects representing the grids for the computer and player.

A screenshot of a computer screen

Description automatically generated

Figure :: SelectionGrid for Player and Computer

* **A screen shot of a computer

  Description automatically generatedaiController:** Manages the computer's moves based on the difficulty.

Figure : aiController

* **A black screen with white text

  Description automatically generatedstatusPanel:** A panel that displays game status and messages.

Figure : statusPanel

c. How it Works:

* The constructor sets up the game panel based on the difficulty level.
* It implements MouseListener and MouseMotionListener to handle player interactions.
* paint method renders the game's graphical components.
* It contains various methods to handle game logic, such as placing ships, firing shots, and processing turns.

d. GamePanel Class Functions:

1. **Constructor GamePanel(GameDifficulty difficulty):**

**A computer screen shot of text

Description automatically generatedPurpose:** Sets up the game panel based on the difficulty.

Figure : GamePanel constructor

**Process:** Initializes grids, AI, and other components necessary for gameplay.

**A screen shot of a computer

Description automatically generatedA screen shot of a computer code

Description automatically generated**

Figure : GamePanel (2)

Figure : GamePanel (1)

**Key Operations:**

* Loading the radar background image.
* Creating player and computer grids.
* Initializing the AI controller based on difficulty.

1. **paint(Graphics g):**

**Purpose:** Renders the game's visual components.

**Process:** Draws the radar background, grids, ships, and status panel.

**Key Operations:**

* Drawing the radar background for both grids.
* Painting the computer and player grids.
* A computer screen with colorful text

  Description automatically generatedHandling the placement and appearance of ships.

Figure : paint method

1. **handleInput(int keyCode):**

**Purpose:** Processes key inputs during the game.

**Process:** Handles various game actions like restarting, rotating ships, and toggling debug mode.

**A screenshot of a computer program

Description automatically generatedKey Operations:** Responding to key events for game controls (like ESC for quit, S for restart, R for rotate, D for activated mode).

Figure : handleInput method

1. **mouseReleased(MouseEvent e) and mouseMoved(MouseEvent e):**

**Purpose:** Handles mouse interactions for placing ships and firing shots.

**Process:** These methods respond to mouse events for ship placement and attacking on grids.

**Key Operations:**

* Updating ship placement based on mouse position.
* *A computer screen with text

  Description automatically generated*Firing at the computer's grid during the attack phase.

Figure : mouseReleased method

**A screen shot of a computer

Description automatically generated**

Figure : mouseMoved method

#### *1.3.8 Graphics-ralated methods*:

Figure : drawRadarBackground method

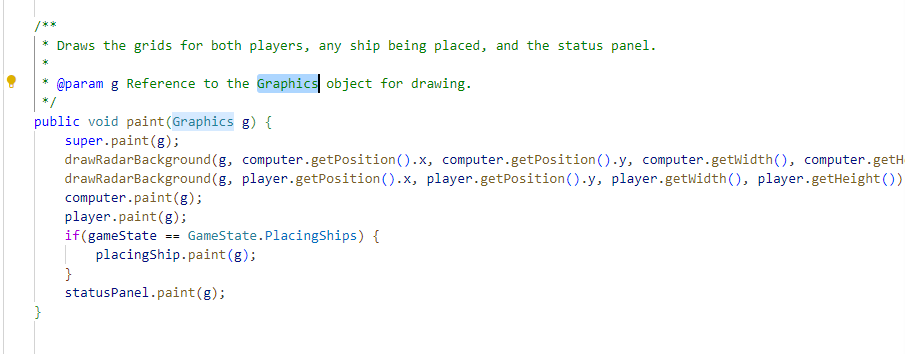
* **Purpose:** The drawRadarBackground method in this code snippet is designed to render a semi-transparent background image (presumably for a radar interface) onto a graphical component.
* **Process:** When invoked with the necessary parameters (graphics context and grid dimensions), it first checks if the radar background image (radarBG) is loaded. It then calculates the size and position for the image to ensure it is centred within the given dimensions, with adjustments for precise placement. Utilizing advanced graphics features, the method casts the Graphics object to Graphics2D for better control and sets a semi-transparent alpha composite. The image is drawn at the calculated position with the specified size and transparency level. Finally, it resets the graphics state to ensure subsequent drawings are unaffected. This method effectively enhances the visual appeal of the radar feature in the GUI.

Figure : paint method in GamePanel class

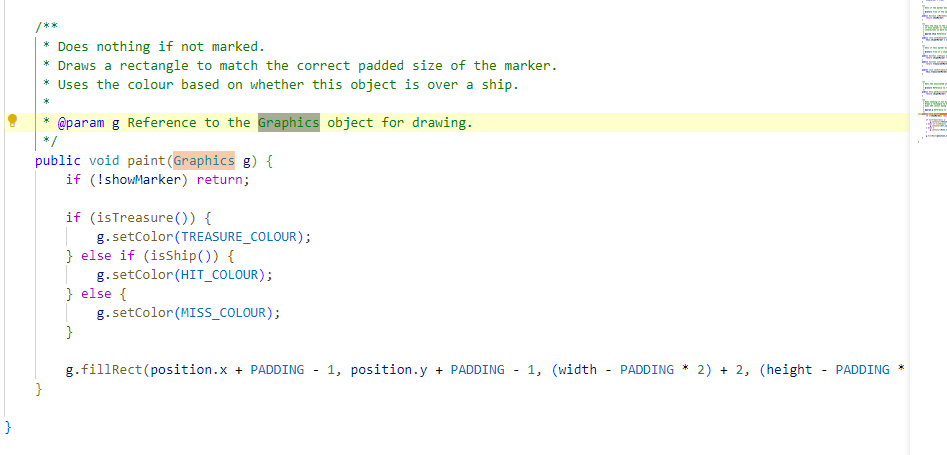
* **Purpose:** The paint method in the GamePanel class serves as the central drawing function for the Battleship game interface. Its primary role is to render the visual components of the game onto the screen. This includes drawing the radar backgrounds for both the player's and the computer's grids, the grids themselves, any ships that are currently being placed during the "PlacingShips" state, and the status panel. The method ensures that all these elements are visually updated in response to game events or player actions, thereby maintaining an interactive and dynamic game environment.
* **Process:** the paint method first calls super.paint(g) to handle any standard painting tasks defined in the superclass (JPanel). It then proceeds to draw the radar background for both the computer's and player's grids by calling drawRadarBackground with the appropriate parameters for each grid. The radar background provides a thematic visual layer behind the grids. Following this, the method calls the paint method of both computer and player objects, which are instances of SelectionGrid. This action renders the grid structures, and any ships or markers present on them. If the game is in the "PlacingShips" state, the method also paints the currently selected ship that the player is positioning. Lastly, the status panel, which displays game messages and status, is drawn through its paint method. This comprehensive painting process ensures that all game components are properly rendered on the screen, reflecting the current state of the game.

Figure : paint method in Marker class

* **Purpose:** The paint method in the Marker class is designed to visually represent the status of each cell in the game grid for the Battleship game. Its primary function is to draw markers on the grid, indicating the results of the player's actions. These markers can signify whether a shot hit a ship, missed, or discovered a treasure. The method enhances the game's interactivity by providing immediate visual feedback on the player's actions, making the gameplay experience more engaging and informative.
* **Process:** The paint method in the Marker class operates by first verifying if the marker should be visible, as indicated by the showMarker flag. If this flag is false, the method terminates without rendering the marker. This step ensures markers are shown only when necessary. If visible, the method assigns a colour based on the marker's state: TREASURE\_COLOUR for treasure, HIT\_COLOUR for a ship, and MISS\_COLOUR otherwise. This colour scheme aids in quickly conveying the game's outcomes. The method then renders a rectangle for the marker, slightly smaller than the cell due to the padding, which helps distinguish it on the grid. These adjustments maintain uniformity in the marker's size. By doing so, the paint method effectively conveys crucial game information, enhancing player engagement.

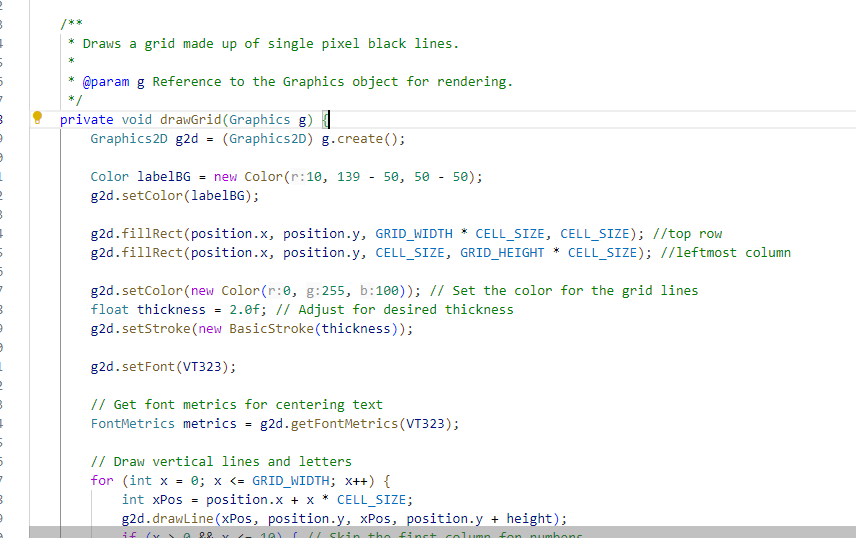
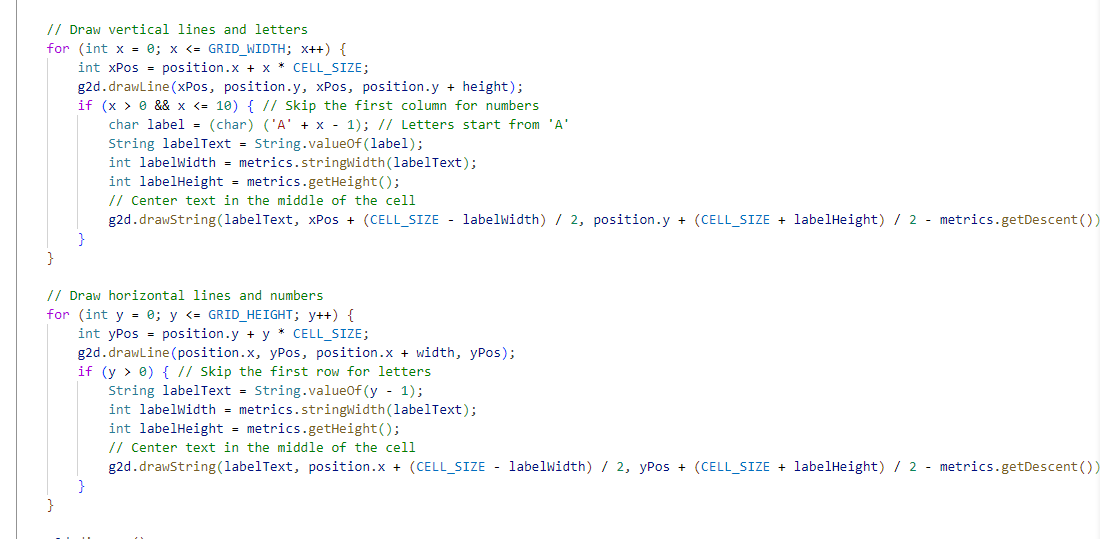


Figure : drawGrid method (2)

Figure : drawGrid method (1)

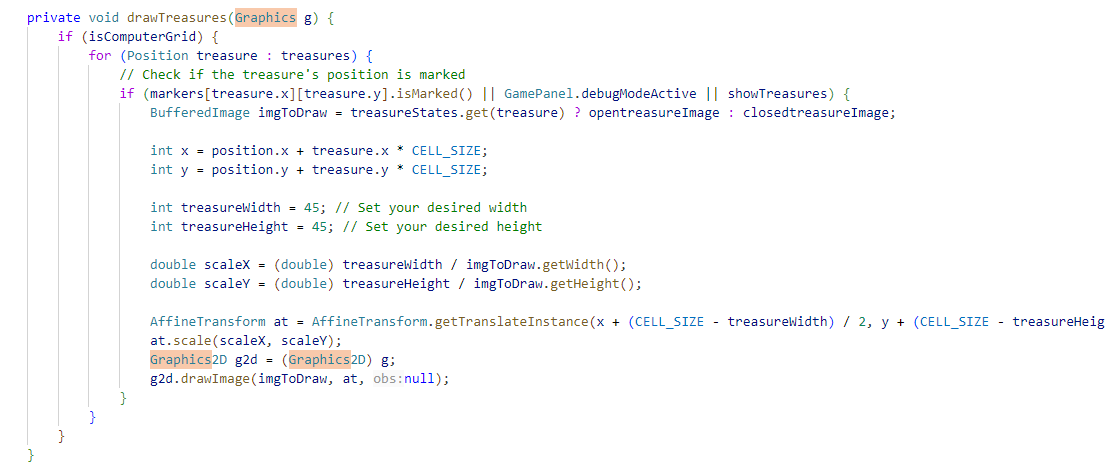
* **Purpose:** The drawGrid method in the provided code snippet is designed to visually render a grid structure using Java's Graphics API. The primary function of this method is to create a grid layout that can be used for various applications such as games, graph plotting, or as a part of a user interface. This method takes a Graphics object as a parameter, which provides the basic drawing utilities. The grid is customized with specific colours, line thicknesses, and text labels. Each cell within the grid is distinctly marked with either a letter (for columns) or a number (for rows), aiding in easy identification and navigation. The method is marked private, indicating that it is intended to be used internally within the class it belongs to, and not directly accessible from other classes.
* **Process:** The drawGrid method begins by creating a Graphics2D object from the passed Graphics object, enabling more sophisticated control over geometry, coordinate transformations, colour management, and text layout. Initially, it sets a background colour for the labels and fills specific rectangles to create the top row and leftmost column of the grid. Then, it sets a colour and stroke thickness for the grid lines. The grid lines, both vertical and horizontal, are drawn using loops that iterate over the grid's width and height. The method uses font metrics to ensure that the text labels (letters for columns and numbers for rows) are centrally aligned in their respective cells. The text is drawn after calculating the appropriate position to achieve this centre alignment. In the case of columns, letters are used as labels starting from 'A', while rows are labelled with numbers starting from 0. After completing the drawing, the Graphics2D object is disposed of to release system resources and ensure efficient memory usage. This method effectively combines graphics drawing techniques to create a user-friendly, visually appealing grid structure.

Figure : drawTreasures method

* **Purpose:** The drawTreasures method is specifically designed to render treasure images on a graphical interface, typically within a game or simulation environment. This method is part of a larger system where it interacts with game elements like treasures, their states, and their positions on a grid. The method takes a Graphics object as its parameter, which is fundamental for drawing images within a Java graphical application. The core purpose of this method is to visually represent treasures in either an open or closed state on the game's grid, depending on certain conditions such as whether the treasure is marked, if the debug mode is active, or if treasures are set to be always shown. This is particularly relevant in scenarios where the grid represents a computer's playing field in a game, adding a layer of interactivity and visual feedback for the player.
* **Process:** Upon invocation, the drawTreasures method first checks whether the grid is designated as the computer's grid through the isComputerGrid boolean flag. If this condition is met, the method iterates over a collection of treasure objects, each representing a treasure's position on the grid. For each treasure, the method determines whether to display an open or closed treasure image. This decision is based on whether the treasure's position is marked, if the game is in debug mode, or if a setting to always show treasures is enabled. Depending on these conditions, the appropriate image (either open or closed treasure) is selected. The method then calculates the x and y coordinates for the treasure's position on the grid, considering the grid's origin (position) and the treasure's position in the grid. It also sets the width and height for the treasure images and calculates scaling factors to ensure that these images are resized appropriately to fit within the grid cells.

#### *1.3.9 Sound Effects:*

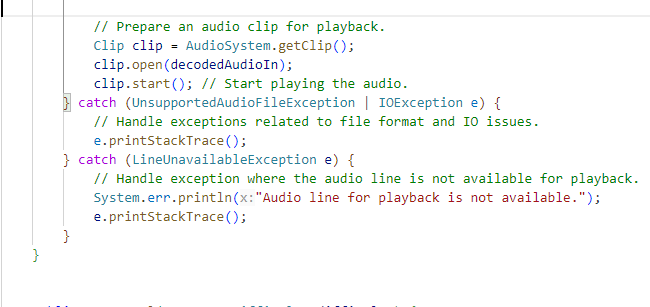
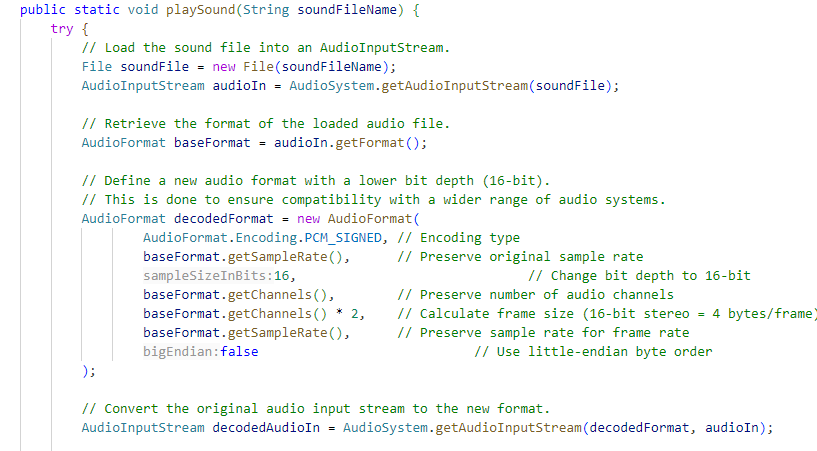
* **Purpose:** Integrating the sound files into the game base on the win and lose condition

Figure : playSound method

Figure : Prepare the audio files

* **Process:** Loading wav files from the codebase to the integrating game logic class Game and GamePanel.

#### A screen shot of a computer code Description automatically generated*1.3.10 Videos:*

Figure : playVideo method (2)

Figure : playVideo method (1)

* **Purpose:** Integrating the video files into the game base on the win and lose condition.
* **Process:** Loading mp4 files from the codebase to the integrating game logic class Game and GamePanel.

## **Diagrams:**

### ***2.1 Behavioural Diagrams:***

#### *2.1.1 Use Case Diagram:*

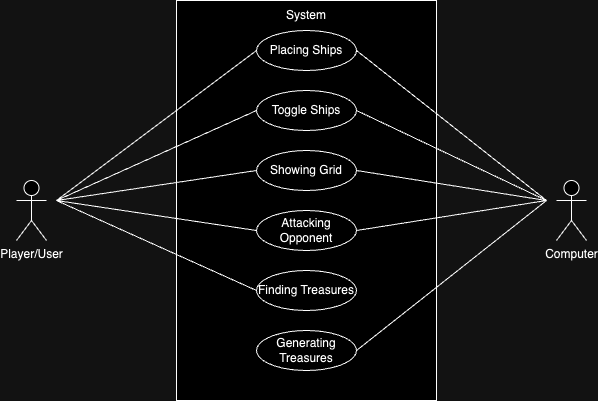
a. Description:

Figure : Use Case Diagram

The Use Case diagram of the Battleship project illustrates the various interactions and functionalities within the system, provides a high-level overview of the system's interactions, showcasing the dynamic exchanges between the primary actors and the essential functionalities that contribute to the overall gameplay experience.

b. Explanation:

* **Actor: Player/User:**
* Placing Ships: The player can strategically place ships on the game grid before the start of the game.
* Toggle Ships: The player can toggle or rearrange the positions of their placed ships for better strategic positioning.
* Showing Grid: This feature allows the player to visualize and review the arrangement of their ships on the game grid.
* Attack Opponent: The player can launch attacks on the opponent's grid by selecting specific positions to target enemy ships.
* Finding Treasures: The player has the objective of discovering hidden treasures within the opponent's grid during the gameplay.
* **Actor: Computer**
* Placing Ships: The computer AI strategically places its ships on the grid as part of the setup for the game.
* Toggle Ships: Like the player, the computer AI can adjust the positions of its ships for strategic purposes.
* Showing Grid: The computer AI has the capability to display its grid to the player, providing insight into its ship positions.
* Attack Opponent: The computer AI engages in attacking the player's grid by selecting specific positions for potential hits.
* Generating Treasures: The computer AI generates hidden treasures within its grid, challenging the player to discover them during the game.

#### *2.1.1 Sequence Diagram:*

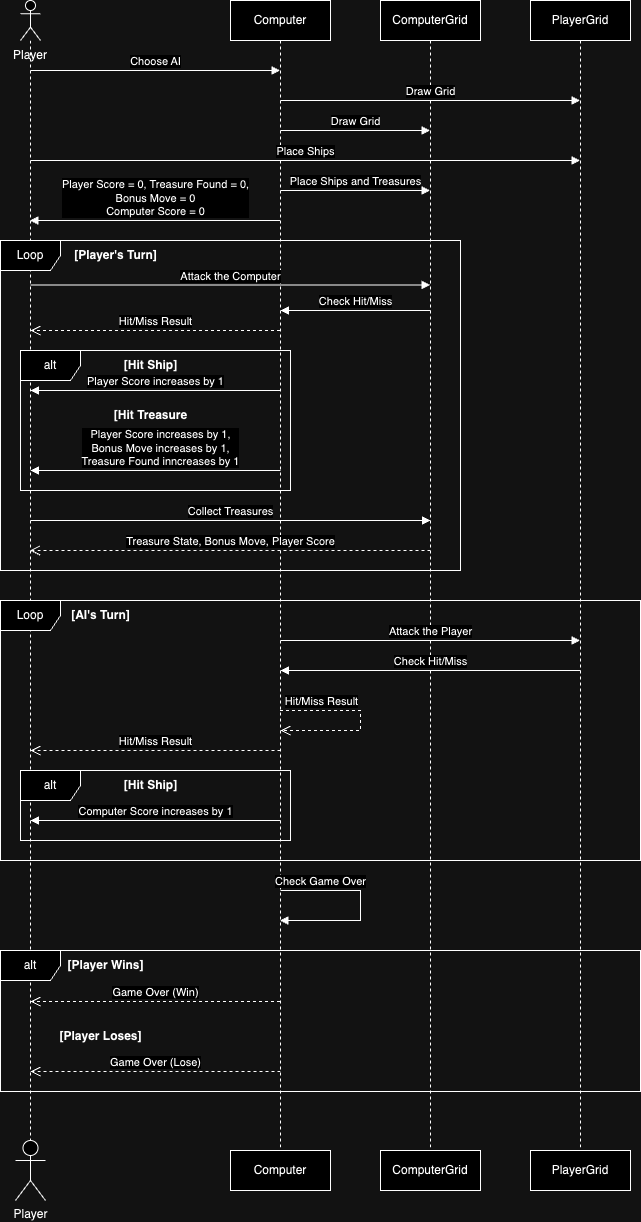


Figure : Sequence Diagram

a. Description:

The sequence diagram encapsulates the dynamic exchange of information and commands between the player and the game system, showcasing the steps involved in ship placement, toggling, and strategic attacks. Additionally, it represents the iterative nature of gameplay, where turns alternate between the player and the computer. The diagram provides a visual representation of the sequential order of events, emphasizing the dynamic and interactive nature of the Battleship game.

b. Explanation:

1. **Player selects AI difficulty:**

Player initiates the game by choosing the difficulty level for the Computer AI.

1. **Computer draw both grids:**

Upon difficulty selection, the computer generates and displays the game grids for both the Computer and the Player.

1. **Player places ships:**

The Player strategically places their ships on the player grid.

1. **Computer places ships and treasures:**

The Computer AI autonomously positions its ships and hides treasures on its own grid.

1. **Computer sends initial game state:**

The Computer communicates the initial game state to the player, including scores (both Player and Computer), Treasure Found, and Bonus Move information.

1. **Player’s Turn Loop:**

* Player attacks Computer on the Computer grid.
* Computer checks for hit/miss of the hit.
* Computer returns hit/miss result to the Player.
* If Player hits ship: The Computer notifies the Player of a successful hit, increase the Player Score by 1.
* If Player hits treasure: The Computer notifies the player of finding treasure, successful hit, increase the Player Score, Bonus Move and Treasure Found by 1.
* Player collects treasures in the Computer grid.
* Computer updates and returns to the Player the information of Treasure State, Bonus Move and the Player Score.

1. **AI’s Turn Loop:**

* Computer attacks Player on the Player grid.
* Computer checks for hit/miss of the hit.
* Computer returns hit/miss results to both it and Player.
* If Computer hits ship: The Computer notifies the Player of its successful hit, increase the Computer Score by 1.

1. **Computer checks for Game Over:**

* The Computer determines whether the game is over.
* If Player wins: The Computer sends a victory message to the Player.
* If Computer wins: The Computer sends a defeat message to the Player.

### ***2.2 Structural Diagrams:***

#### *2.2.1 UML Diagram (Class Diagram):*

Figure : UML Diagram

a. Description:

The UML diagram for the Battleship Game provides a comprehensive visual representation of the system's structure and behaviour. The diagram includes various classes, components, and relationships crucial for understanding the game's architecture.

b. Explanation:

* **BackgroundPanel class:** A custom JPanel designed to display an image as a background and animate text on top of it.
* **BattleshipAI class:** A template class exists which can be extended to design the actual behaviour of the AI. There is a method called selectMove() that helps determine the move to be applied during gameplay. Classes that aim to introduce real functionality should override this selectMove() method.
* **ButtonManager class:** Manages custom buttons and border styles for Swing components.
* **Game class:** The Game class initializes the main frame and controls the navigation between different panels representing game states like difficulty selection and strategy display.
* **GamePanel class:** It controls the state information and interactions among game elements, overseeing two game grids: one for the human player and another for the computer, separated by a gap. This panel displays the current game state, allowing players to place their ships and target their opponent. During ship placement, the player can position their ships on their grid. In the attack phase, the player targets the computer's grid to destroy its ships and find the hidden treasures. The status panel provides updates on game progress and the comparative state of both player and computer grids. It also handles player inputs, enabling actions like ship placement, attacking, and toggling debug mode. The AI determines the computer's moves, and the panel manages the game flow based on the current state. Additionally, it contains methods to handle mouse and key events for gameplay interactions.
* **Marker class:** Represents a simple coloured rectangle that can be either shown or hidden and will change colour based on whether it is representing the location where a ship is.
* **PlaySound class:** The PlaySound class provides a utility method to play audio files using Java's Sound API.
* **PlayVideo class:** The PlayVideo class facilitates playing video files using JavaFX in a Swing environment.
* **Position class:** This class is used to represent a single position x, y.
* **Rectangle class:** Defines a simple Rectangle with a position for the top left corner, and a width/height to represent the size of the Rectangle.
* **SelectionGrid class:** Represents a grid for the game, managing ships, treasures, and markers for gameplay.
* **Ship class:** Defines a simple ship object that can be drawn onto the screen. Stores information about how many segments it includes, the direction of the ship, how many of the segments have been destroyed, and its state to set the colour.
* **SimpleRandomAI class:** A very simplistic AI that does not use any common-sense. It will just shuffle the list of valid moves into a random order, and then select moves based on the ones that appear first in the list.
* **SmarterAI class:** Defines an AI that will search randomly until it finds ships. Then attempts to attack cells around the discovered location to ruthlessly go after ships.
* **StatusPanel class:** Defines a simple text panel to show a top and bottom line of text. Some of these are already defined in the class, and it provides additional methods to set the messages to custom values and display the score system of the game.

#### *2.2.2 Entity Relationship Diagram (ERD):*

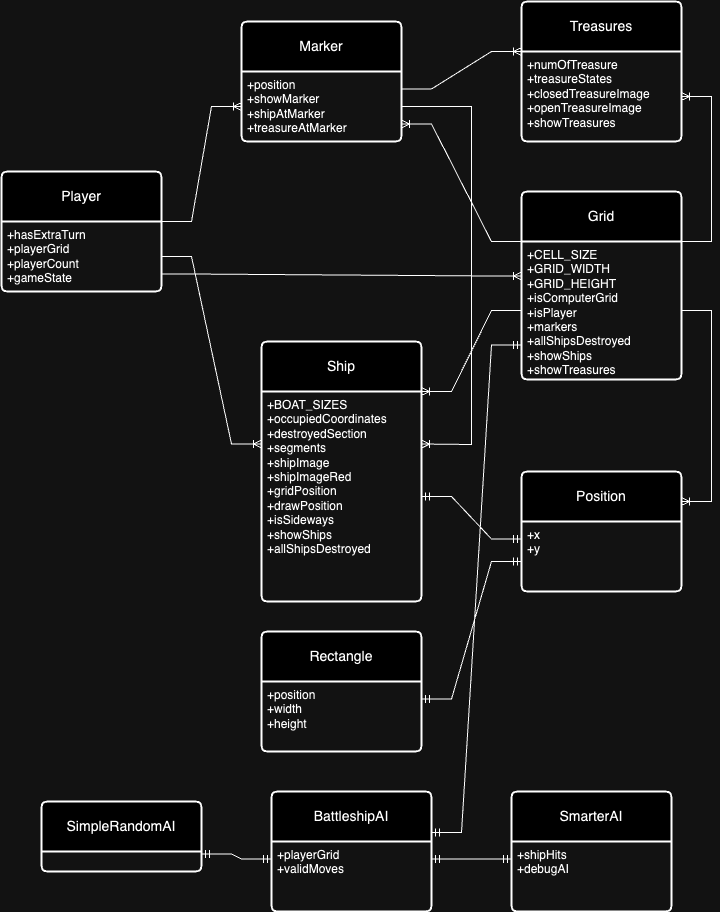
a. Description:

Figure : ERD Diagram

The Entity-Relationship Diagram (ERD) for the Battleship Game illustrates the key entities and relationships within the game's data model. The ERD captures the essential aspects of the game's structure, emphasizing how different entities are connected and interact.

b. Explanation:

* **Entities:**
* Player: Represents the game player and stores attributes like hasExtraTurn, playerGrid, playerCount, gameState.
* Ship: Represents individual ships in the game, including some important attributes like BOAT\_SIZES, occupiedCoordinates, destroyedSection, segments, gridPosition, drawPosiotion, showShips, allShipsDestroyed.
* Treasure: Represents hidden treasures within the game, including numOfTreasure, treasureStates.
* Grid: Represents the game grids, containing information about CELL\_SIZE, GRID\_WIDTH, GRID\_HEIGHT, isComputerGrid, isPlayer, markers, allShipDestroyed, showShips, showTreasures.
* Position: Represents the position
* BattleshipAI: Represents the overarching entity for different AI implementations.
* SimpleRandomAi:Represents a simple random AI strategy.
* SmarterAI: Represents a more advanced AI, which can select moves more strategically and have attributes like shipHits to serve its algorithm.
* **Relationships:**
* Player-Grid relationship (One-to-Many): A player can have multiple grids, but each grid is associated with only one player. This relationship represents the ownership of grids by players.
* Player-Ship relationship (One-to-Many): A player can own multiple ships, but each ship is owned by only one player. This relationship reflects the ownership of ships by players.
* Player-Marker relationship (One-to-Many): Each player can have multiple markers. Markers could represent various in-game indicators or annotations associated with a player, providing additional information.
* Grid-Marker relationship (One-to-Many): Each grid can have multiple markers. Like the Player-Marker relationship, this association allows grids to be annotated with various indicators or information.
* Grid-Ship relationship (One-to-Many): Each grid can have multiple ships, but each ship is located on only one grid. This relationship signifies the placement of ships on a specific grid.
* Grid-Treasure relationship (One-to-Many): Each grid can have multiple treasures, indicating that treasures are associated with specific grids. This relationship represents the distribution of treasures across the game grids.
* Ship-Position relationship (One-to-One): Each ship has a unique position, and each position is associated with only one ship. This one-to-one relationship denotes the specific position of a ship on the grid.
* BattleshipAI-Grid relationship (One-to-One): Each BattleshipAI is associated with one grid. This relationship indicates that the AI strategy is applied to a specific game grid.

### ***2.3 Swimlane Diagram:***

a. Description:

A Swimlane Diagram for the Battleship Game illustrates the flow of activities and responsibilities among different entities or participants involved in the game.

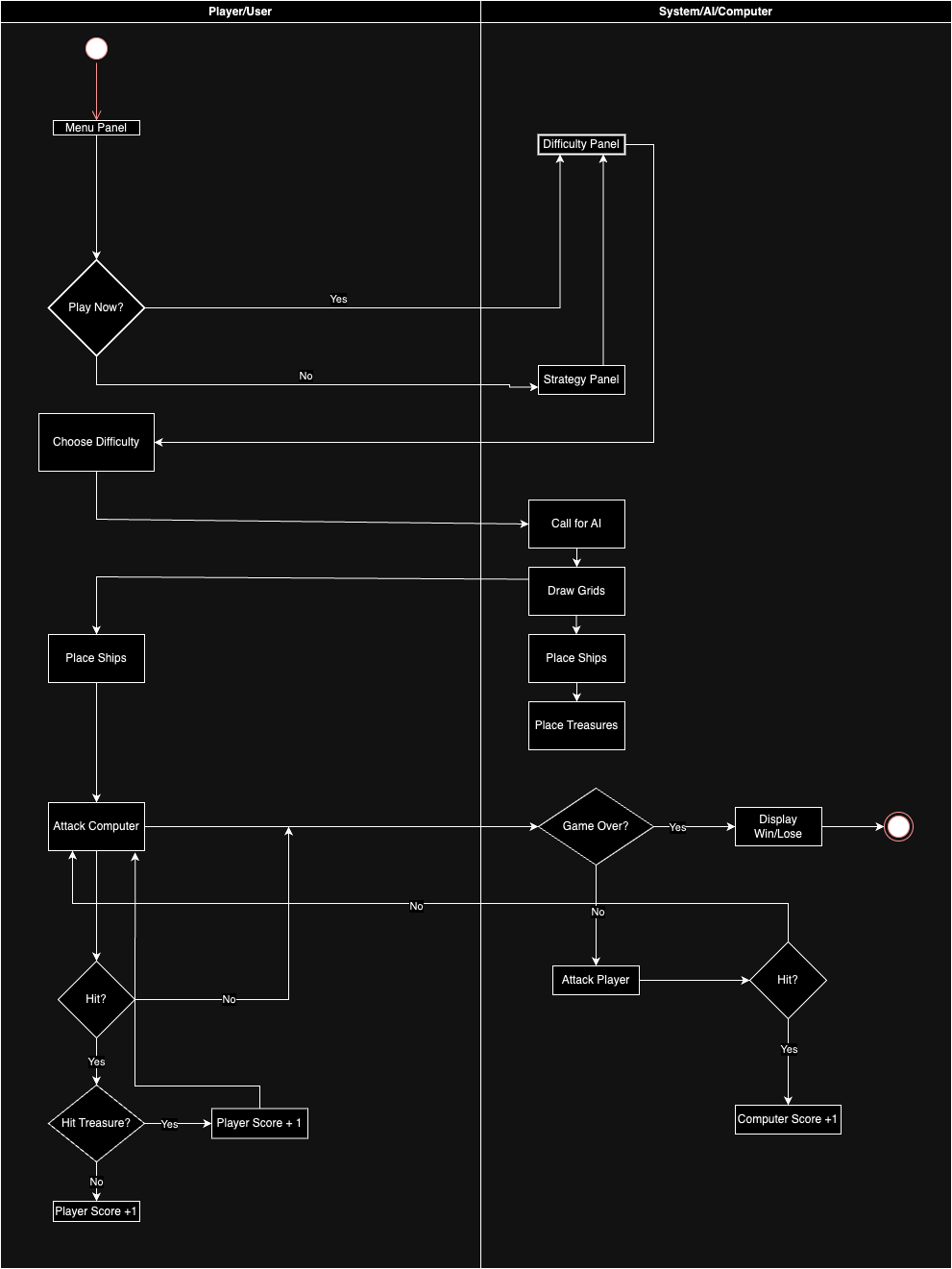


Figure : Swimlane Diagram

b. Explanation:

* **Player/User:**
* Responsibilities: Placing ships on the grid, toggling ship positions, attacking opponents by selecting grid coordinates, and interacting with the game interface.
* Interaction: Communicates with the player grid, ships, treasures, and opponent grid.
* **Computer/System:**
* Responsibilities: Placing ships and treasures on the grid, toggling ship positions, attacking opponents through AI-driven strategies, managing its own game grid, calculate and display the scoring system and manage the whole game logics.
* Interaction:
* Interacts with the game grid, ships, and opponent grid using AI algorithms.
* Receives commands from both the Player and Computer, updates ship positions, and provides feedback on attacks.

### ***2.4 System Diagrams:***

#### *2.4.1 Technology Diagram*:

Figure : Technology Diagram

a. Description:

The Battleship Game is developed using Java for both the graphical interface and game logic. The integrated development environments (IDEs) include VSCode and IntelliJ IDEA, while version control is managed through GitHub and Git. Task management is organized using Notion, and communication among team members is facilitated through messaging platforms. This technology stack ensures efficient development, collaboration, and version control throughout the project lifecycle.

b. Explanation:

* **Graphical Interface/Frontend:** Java was chosen for the Battleship Game's graphical user interface (GUI) due to its platform independence, robust libraries like Swing and JavaFX for interactive design, and strong community support, facilitating development, maintenance, and scalability.
* **Game Logics/Backend:** Chosen as the backend programming language. Java is known for its portability, scalability, and robustness so we decided to choose it for building enterprise-level applications.
* **IDE:** Visual Studio Code (VSCode) and IntelliJ IDEA were chosen as the Integrated Development Environments (IDEs) for the Battleship Game project due to their user-friendly interfaces, extensive plugin ecosystems, and powerful features that enhance code editing, debugging, and version control. These IDEs offer excellent support for Java development, contributing to increased productivity and code quality throughout the project lifecycle.
* **Version Control:** GitHub is a version control platform that facilitates collaborative development and code management. It allows us to work together on the same project, track changes, and manage different versions of the codebase.
* **Task Management:** Notion is the tool that is used for task management and collaboration. We use this tool to help our team organize, track, and manage member’s work. Messenger is the platform served as the mean of communication of all the team.

#### *A black and white diagram Description automatically generated2.4.2 Architecture Diagram:*

Figure : Architecture Diagram

a. Description:

The architecture diagram provides a high-level overview of the Battleship Game's modular structure, showcasing the key components and their interactions. It serves as a visual guide to understand how different modules, such as graphical interfaces, game logics, audio, video, and file system management, contribute to the overall functionality of the game. This diagram aids developers and stakeholders in comprehending the system's organization, facilitating efficient collaboration, and enabling targeted discussions about specific components.

b. Explanation:

* **Graphical Interfaces:**
* Role: Manages user interface and interactions.
* Functionality: Facilitates user input and forwards it to the game logics module.
* **Game Logics:**
* Role: Core module handling game mechanics.
* Functionality:
* Ship Placement.
* Win/Lose Conditions.
* Treasure Logics.
* Scoring.
* Feature: Real-time score tracking during gameplay.
* **Audio Module:**
* Role: Manages game audio elements.
* Functionality:
* Playing shooting sounds.
* Playing win/lose sounds.
* Utilizes a sound player for audio playback.
* **Video Module:**
* Role: Manages game video elements.
* Functionality:
* Handling win video.
* Handling lose video.
* Utilizes a video player for video playback.
* **File System Module:**
* Role: Handles resources loading from the system.
* Functionality:
* Loading fonts.
* Loading images.
* Loading sounds.
* Loading videos.

## **Key Designs:**

The essence of the key design in the Battleship Game revolves around core entities like ships, treasures, grids, and panels, coupled with the introduction of diverse levels. The Ship Design introduces strategic naval elements, while Treasures adds an exciting layer of surprise within the grids. Grid Design forms the battleground, organizing cells for gameplay. The inclusion of Levels Design enriches the experience, providing varying degrees of difficulty by calling different AI for each difficulty level. This amalgamation creates a dynamic and visually engaging Battleship Game, seamlessly blending entities and levels for an immersive gaming journey.

### ***3.1 Entities:***

The core entities of the Battleship Game form the backbone of its structural design, playing pivotal roles in the game dynamics and overall user experience. This section delves into the key design entities, namely ships, treasures, grids, and panels, offering insights into their individual significance and collective contribution to the immersive gameplay.

#### *3.1.1 Grids:*

The game encompasses two distinct grids: the player grid and the computer grid. Each grid consists of 11 rows and 11 columns, with one row and one column designated as labels where ships or treasures cannot be placed. The backdrop of these grids features the visually impactful radar image, a key element of the graphical user interface (GUI). During gameplay, when both participants are engaged, the markers on the grid dynamically change their appearance to indicate hits or the discovery of treasures. This interactive visual feedback system enhances the player's engagement and understanding of the ongoing naval battle, making the GUI not only functional but also aesthetically enjoyable.

Below the grid, two essential components contribute to the overall game interface: the score banner and the status panel. The score banner serves as a dynamic tracker for hit points, providing players with real-time feedback on their progress in the game. Meanwhile, the status panel plays a crucial role in communicating various messages and instructions, adapting its content based on the current state or specific conditions within the game's underlying logic. This dual-panel setup not only enhances the user experience by presenting key information briefly but also ensures that players receive relevant guidance and updates throughout different stages of the gameplay.

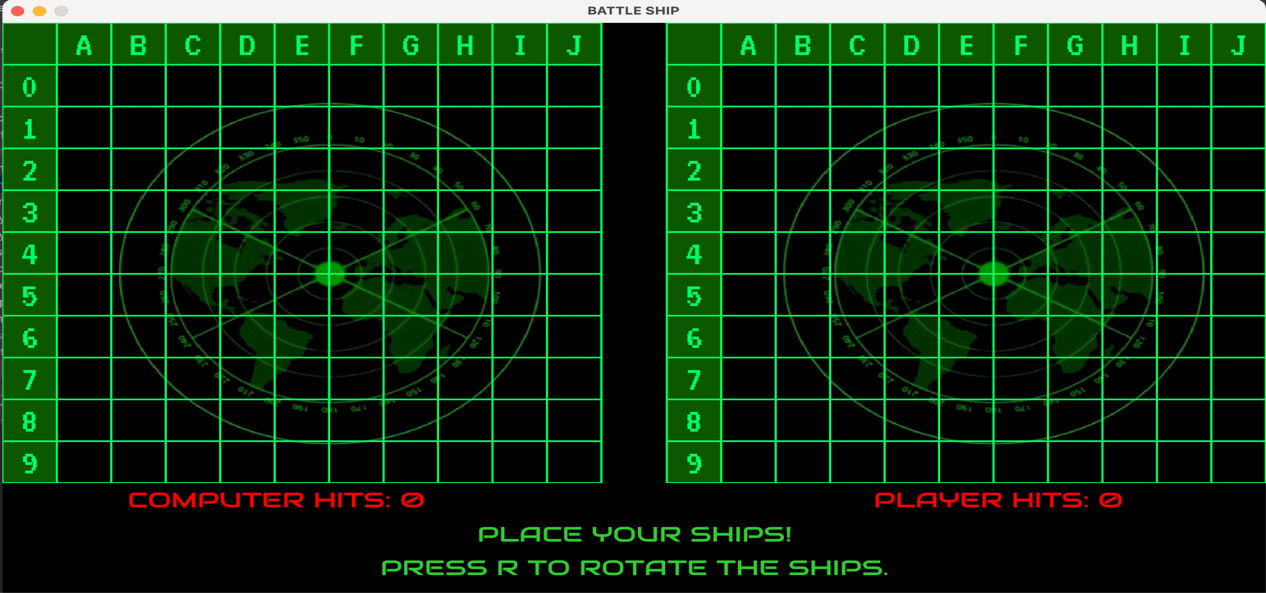


Figure : Game Window with Grids

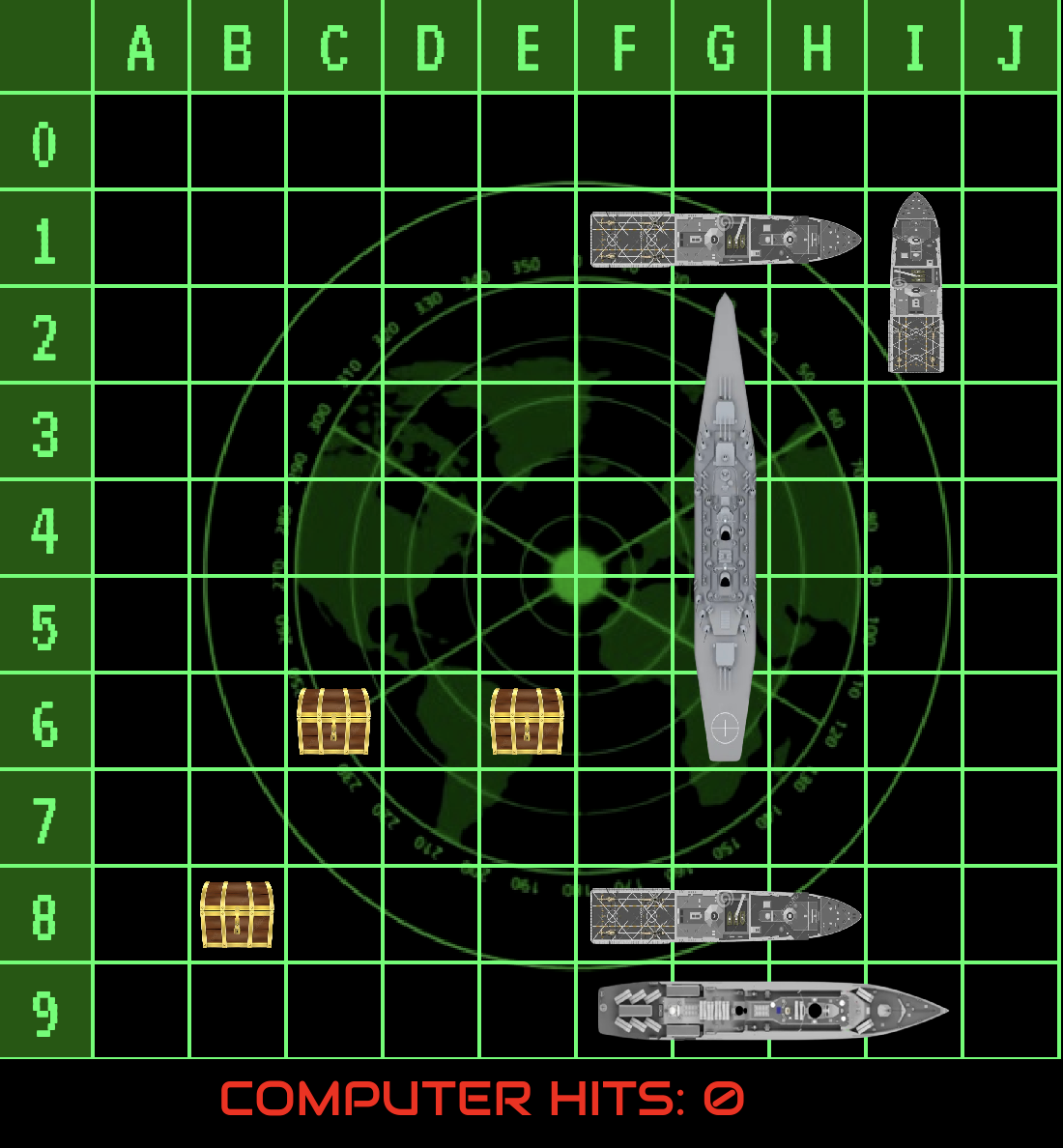


Figure : Computer Grid with Ships and Treasures in Debugmode

#### *3.1.2 Ships:*

The Ship entity plays a crucial role in our Battleship Game structure, capturing the spirit of naval warfare in the digital domain. This entity encompasses different ship varieties, distinguished by distinct sizes and strategic importance on the gaming grid. Ships play a central role in both offensive and defensive game phases, being involved in the placement stage and serving as targets during attacks.

The fleet comprises ships of four distinct sizes: 5 segments, 4 segments, 3 segments (with two variations), and 2 segments. Each ship size is associated with a unique image.

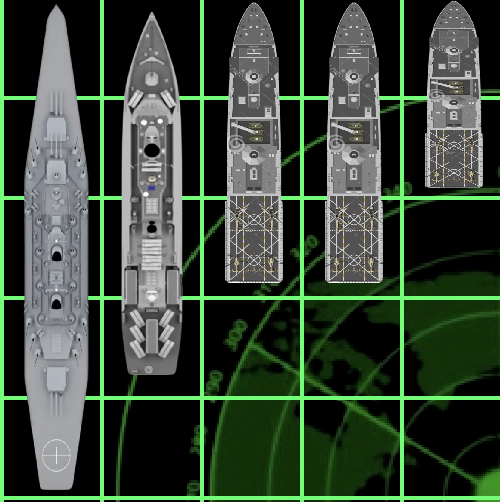
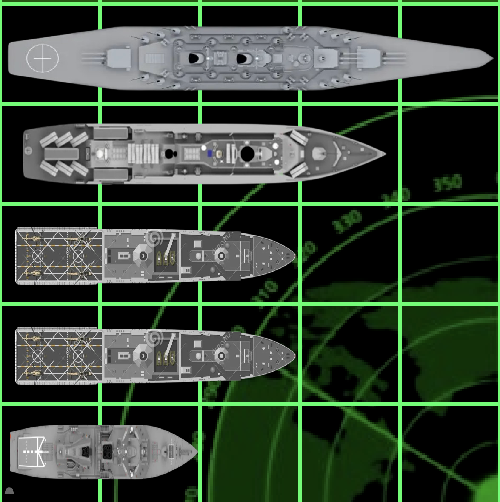


Figure : Ships vertically

Figure : Ships horizontally

When a ship is successfully sunk during gameplay, its corresponding image undergoes a transformation, featuring a red filter layer to visually signify its sunken status. This mechanism adds an extra layer of visual feedback, enhancing the gaming experience by clearly marking the fate of each ship on the grid.

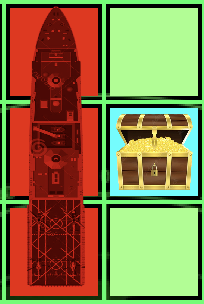


Figure : Marker on Hit, Miss, Treasure and Sunk Ship

#### *3.1.3 Treasures:*

Within the game mechanics, three hidden treasures are strategically placed at random positions on the computer's grid by the system. These treasures remain concealed, depicted by a closed image.



Figure : Closed Treasure

Upon successfully hitting a position containing one of these treasures, the image transforms to an open state accompanied by a distinctive sound, signalling the player's discovery of the hidden treasure. This element adds an exciting and interactive dimension to the gameplay, rewarding players with both visual and auditory cues upon uncovering these elusive items.



Figure : Opened Treasure

#### *3.1.4 Game Panels:*

a. Menu Panel:

The Menu Panel serves as the initial interface that players encounter upon starting the game. It presents three distinct option buttons to the player:

* **"PLAY NOW":** This option is designed for players who are familiar with the game rules and are ready to commence gameplay immediately.
* **"STRATEGY":** Geared towards new players, this option provides guidance and strategies to help newcomers understand the game better before diving into gameplay.
* **"EXIT":** For those who wish to close the game window, the "Exit" option allows players to gracefully exit the game.



Figure : Menu Panel

b. Strategy Panel:

The Strategy Panel is accessed by clicking the "STRATEGY" button in the menu panel, providing players with succinct instructions for the game. Through an animated method, the content of the strategy is dynamically presented against the backdrop image, creating an immersive and visually appealing experience. This approach not only enhances engagement but also contributes to a seamless integration of game instructions with the overall design.

The Strategy Panel also features a "START" button, allowing players to transition seamlessly from the instructional phase to the actual gameplay. This user-friendly interface aims to facilitate a smooth and enjoyable experience for players, combining animation and clear layout to enhance comprehension and engagement.



Figure : Strategy Panel

c. Difficulty Panel:

The Difficulty Panel is activated when players click the "START" button on the Strategy Panel or the "PLAY NOW" button on the Menu Panel. This crucial component of the game interface presents players with a choice of three difficulty levels:

* **"EASY":** For newcomers.
* **"MEDIUM":** For those familiar with the game.
* **"HARD":** For players seeking a challenging experience.

Each difficulty level is accompanied by a concise description outlining the characteristics of the respective artificial intelligence (AI). This informative display assists players in making an informed decision about the level of challenge they wish to undertake. By selecting one of the three buttons corresponding to the desired difficulty, players can seamlessly progress into the gameplay phase, ensuring a tailored and enjoyable experience for individuals with varying skill levels.



Figure : Difficulty Panel

### ***3.2 Levels:***

In the game, three distinct levels are linked to three corresponding Artificial Intelligences (AIs). The player's choice of difficulty level determines which specific AI will be invoked to participate in the gameplay.

#### *3.2.1 Easy:*

The easy level introduces a Simple Random AI, characterized by its straightforward approach to gameplay. This AI makes random moves from a predefined list of valid positions, devoid of intricate algorithms. Tailored for players seeking a gentle introduction to the game, the Simple Random AI provides a user-friendly experience by focusing on simplicity and unpredictability. This uncomplicated strategy ensures that newcomers can grasp the fundamentals of the game without contending with complex decision-making processes.

#### *3.2.2 Medium:*

The medium level introduces the Smarter AI, adding a layer of sophistication to the gameplay experience. This AI exhibits a more advanced strategy by favouring grid positions with a higher density of unattacked points during randomized attacks. Designed for players familiar with the game seeking a moderate challenge, the Smarter AI strikes a balance between unpredictability and strategic decision-making. This level provides an opportunity for players to further hone their skills, introducing a nuanced approach to gameplay while maintaining a manageable level of complexity.

#### *3.2.3 Difficulty:*

In the hard level, the game challenges players with the inclusion of the Smarter AI, further enhanced by a strategic attack plan. While the AI maintains its preference for grid positions with a higher concentration of unattacked points, it introduces a heightened level of sophistication when targeting enemy ships. Upon successfully hitting a ship, the AI adapts by strategically choosing subsequent moves that are likely to form a line with the initial hit position. This advanced feature adds a layer of complexity, requiring players to anticipate and counter the AI's strategic responses. The hard level is tailored for seasoned players seeking an intense and strategic gameplay experience, pushing the boundaries of their tactical skills within the digital naval battlefield.

# **CHAPTER IV: DEVELOPMENT**

## **Development Process’s Description:**

### ***1.1 Ideas Collecting:***

The development of project was methodically conducted in three principal phases which were marked by specific methodologies and collaborative efforts. The initial phase centred around the collection and assessment of ideas. Here, a range of insights was gathered from team members, rigorously evaluated for their fit with the project's framework and objectives. This stage culminated in the selection of the most feasible and relevant ideas, followed by a strategic allocation of tasks to efficiently bring these concepts to fruition.

### ***1.2 Code Process:***

In the coding phase, individual team members engaged in the development of code, utilizing their preferred Integrated Development Environments (IDEs). This critical stage involved translating the selected ideas into practical code segments. Each contribution, upon completion, was integrated into the project's GitHub repository. Prior to this integration, a vital quality control step was undertaken by the project manager, who meticulously reviewed each code submission. This review process was pivotal in ensuring adherence to coding standards, consistency with the overall project goals, and compatibility with the existing code framework. The project manager's approval was essential for the incorporation of code into the main project branch, thereby maintaining the integrity and efficiency of the project.

### ***1.3 Bug Fixing – Testing:***

The final stage, encompassing bug-fixing and testing, involved a proactive approach to identifying and resolving any errors that emerged during the earlier phases. Team members contributed significantly by sharing their encounters with bugs, facilitating a swift and transparent resolution process through the GitHub platform. In instances where immediate solutions to coding challenges were not apparent, a more collaborative problem-solving approach was adopted. Team members, including code owners, engaged with the project manager and their peers to collectively address these challenges, leveraging the diverse expertise within the group. This phase was not only crucial in ensuring a timely resolution of issues but also instrumental in fostering a strong sense of collaboration within the team, thereby enhancing the overall quality and reliability of the project.

## **Programming languages, Frameworks and Tools:**

### ***2.1. IDE:***

VSCode and IntelliJ IDEA were chosen to be the main IDEs of the project.

### ***2.2. Language:***

Java was chosen to be the main and only programming language of the project.

### ***2.3. Support:***

Library: To generate the video-effect, the javaFx library was set in the code base.

## **Challenges:**

The initial concept involved the incorporation of an auditory feature, specifically, the emission of a shooting sound each time a player interacted with a specific rectangular element within the game. However, this seemingly straightforward idea was met with a significant obstacle. The developers encountered numerous bugs when attempting to access the sound library, which hindered the implementation of the desired feature.

In a similar vein, the integration of video effects into the game presented its own set of difficulties. The game’s codebase required intricate configuration to accommodate these effects, a task that proved to be particularly challenging. This complexity was further exacerbated by the fact that the game was being developed using two distinct tools, namely VScode and IntelliJ IDEA.

Moreover, the code was being modified across two different operating systems: Windows and MacOS. This introduced an additional layer of complexity when the developers needed to exchange code and communicate about the features. The discrepancies between the two operating systems made it challenging to maintain consistency and ensure seamless integration of the code.

## **Code Management and Version Control:**

Code management and version control played a key role in our project. The main tools used for this purpose are GitHub for collaboration and version control and Messenger for team communication. This report provides an overview of how these tools are used and their impact on our project development.

### ***4.1 Version Control with Git:***

GitHub serves as our primary version control system, using the Git distributed version control system. This allows us to track changes made to the codebase, maintain a comprehensive history, and facilitate collaboration between team members. Git's branching and merging capabilities allow for the concurrent development of features, ensuring a smooth and organised development process.

### ***4.2 Collaborative Workflow:***

GitHub's collaboration features, including pull requests, issues, and project boards, have been instrumental in streamlining our collaborative workflow. Pull requests provide a structured process for code review, allowing our team to review proposed changes before merging them into the main codebase. Issues and project boards help in tracking tasks, bugs, and project milestones.

### ***4.3 Contributor Management:***

GitHub also provides robust tools for contributor management. Each of our team members has their own GitHub account, and access to repositories is controlled through permissions. This ensures a secure and controlled environment, allowing us to manage contributions effectively and maintain code quality.

### ***4.4 Group Chat on Messenger:***

Messenger serves as our primary communication platform, providing an instant messaging solution for team members. The group chat on Messenger facilitates quick communication, updates on project progress, and coordination among team members. It serves as a centralised communication hub for both technical and non-technical discussions.

# **CHAPTER V: TESTING AND BUG TRACKING**

## **Testing Methodologies:**

The development of the Battleship game leveraged Agile methodologies to enhance flexibility and rapid iteration. Key aspects included:

* Scrum Framework: The project was divided into short sprints, each focusing on a specific set of features, allowing for rapid feedback and adjustments.
* Daily Stand-ups: Regular meetings were held to assess progress and address any immediate challenges.
* Task Tracking: A Kanban board was utilized to monitor task progress and individual responsibilities.
* Sprint Retrospectives: At the end of each sprint, the team reviewed accomplishments and challenges to continually improve the development process.

## **Results and Bug Tracking:**

Several test cases were designed to ensure the functionality and reliability of the Battleship game.

|  |  |
| --- | --- |
| **Test Case #:** 1 | **Test Case Name:** Grid Functionality Testing |
| **Designed by:** Mr. Nguyen Phat. | **Design Date:** 8/1/2024. |
| **Executed by:** Mr. Pham Nguyen Dang Khoi. | **Execution Date:** 9/1/2024. |
| **Short Description:** To ensure that the game grid correctly registers and displays hits, misses, and ship placements.  **Result:**  Passed, with all functionalities performing as expected. | |

Table 2: Test Case 1

|  |  |
| --- | --- |
| **Test Case #:** 2 | **Test Case Name:** AI Behaviour Testing |
| **Designed by:** Mr. Nguyen Phat. | **Design Date:** 8/1/2024. |
| **Executed by:** Mr. Pham Nguyen Dang Khoi. | **Execution Date:** 15/11/2023. |
| **Short Description:** Testing various difficulty levels of AI to ensure they behave according to their designed strategies.  **Result:** Passed, but some improvements suggested for Hard AI in strategic edge case scenarios. | |

Table 3: Test Case 2

|  |  |
| --- | --- |
| **Test Case #:** 3 | **Test Case Name:** User Interface Responsiveness |
| **Designed by:** Mr. Nguyen Phat. | **Design Date:** 8/1/2024. |
| **Executed by:** Ms. Huynh Ngoc Anh Thu. | **Execution Date:** 9/1/2024. |
| **Short Description:** To test the game's user interface responsiveness and adaptability across different devices and screen sizes. | |
| **Result:** Passed, the game displayed correctly on various devices (Win, MAC) with no loss of functionality or visual distortion. | |

Table 4: Test Case 3

|  |  |
| --- | --- |
| **Test Case #:** 4 | **Test Case Name:** Sound and Music Integration |
| **Designed by:** Mr. Nguyen Phat. | **Design Date:** 8/1/2024. |
| **Executed by:** Mr. Pham Nguyen Dang Khoi. | **Execution Date:** 9/1/2024. |
| **Short Description:** To verify the proper functioning of in-game sound effects and background music. | |
| **Result:** Passed, all sound effects and music tracks played correctly at appropriate times and levels. | |

Table 5: Test Case 4

|  |  |
| --- | --- |
| **Test Case #:** 5 | **Test Case Name:** Difficulty Level Accuracy |
| **Designed by:** Mr. Pham Nguyen Dang Khoi. | **Design Date:** 8/1/2024. |
| **Executed by:** Ms. Huynh Ngoc Anh Thu. | **Execution Date:** 9/1/2024. |
| **Short Description:** To ensure that the game's difficulty levels (Easy, Medium, Hard) reflect noticeable differences in AI behaviour. | |
| **Result:** Passed, each difficulty level presented a distinct challenge, with the AI exhibiting progressively smarter strategies. | |

Table 6: Test Case 5

|  |  |
| --- | --- |
| **Test Case #:** 6 | **Test Case Name:** Performance Under Load |
| **Designed by:** Mr. Nguyen Phat. | **Design Date:** 8/1/2024. |
| **Executed by:** Ms. Huynh Ngoc Anh Thu. | **Execution Date:** 9/1/2024. |
| **Short Description:** To assess the game's performance under high load conditions, such as rapid consecutive moves. | |
| **Result:** Fail, the game exhibited significant lag under heavy load | |

Table 7: Test Case 6

## **Quality Assurance Measures:**

### ***3.1 Bug-Tracking Process:***

#### *3.1.1 Bug Identification:*

* During testing phases, any anomalies, glitches, or deviations from expected behaviour are documented as bugs.
* Testers use a standard format for reporting bugs, including a clear description, steps to reproduce the issue, expected vs. actual results, and screenshots if applicable.

#### *3.1.2. Regression Testing:*

* After a set of bugs is fixed, regression testing is conducted to ensure that new changes have not introduced additional issues.

### ***3.2 Quality Assurance Measures:***

#### *3.2.1. Test Coverage:*

Ensuring comprehensive test coverage that includes all functional areas of the game, as well as edge cases and failure scenarios.

#### *3.2.2. Performance Metrics:*

Monitoring key performance metrics such as load times, responsiveness, and resource utilization under different conditions.

#### *3.2.3. Code Reviews:*

Regular code reviews to maintain code quality, adherence to coding standards, and early detection of potential issues.

#### *3.2.4. Compliance Checks:*

Ensuring the game complies with relevant legal and platform-specific requirements and standards.

# **CHAPTER VI: GAME INSTRUCTION**

Welcome to the digital rendition of the Battleship game, an innovative adaptation showcasing naval warfare strategies against an AI opponent across distinct difficulty levels: Easy, Medium, and Hard. Navigating through the gameplay involves several key components:

## **Begin the Game:**

* Run the Game.java.
* Menu Panel:

Commence your experience at the Menu Panel, presenting three options: "Play Now" for those acquainted with the rules, "Strategy" catering to newcomers, and "Exit" to conclude the gaming session.

* Strategy Panel:

Activation of the "Strategy" button reveals a comprehensive instructional display. The content unfolds with an animation, providing insights. Initiate gameplay by selecting "Start."

* Difficulty Panel:

Selection of a difficulty level on the Difficulty Panel initiates the game: "Easy" for novice players, "Medium" for an intermediate challenge, or "Hard" for a strategic test.

## **Let's play!**

* Game Grids:

Encounter two grids, each featuring 11 rows and columns, labelled on the first row and column. Your grid hosts ships and treasures, while the computer's treasures remain concealed. The grid background showcases a radar image.

* Markers and Scores:

Markers discern hits and treasures during gameplay. The Score Banner monitors hit points, while the Status Panel conveys contextual messages. Accumulate points by sinking the computer's ships and discovering treasures.

* Treasures:

Uncover three hidden treasures on the computer's grid. A closed image transforms into an open one upon discovery, accompanied by an auditory confirmation.

* Gameplay:

Employ strategic attacks to locate and sink the computer's ships. Tailor tactics according to the chosen difficulty level, each characterized by distinct AI behaviour.

## **Victory or Defeated?**

* Victory:

Attain victory by successfully sinking the computer's ships and revealing treasures. Tactical acumen will prove pivotal in conquering the digital naval arena.

* Defeated:

If the AI destroys all your ships before you do, you are defeated. Press S to restart the game and challenge the computer again.

# **CHAPTER VII: LESSONS LEARNED**

## **Lesson Learned: Project Monitoring and Task Management**

Throughout the course of our project, we gained valuable insights into effective project monitoring and task management. The use of tools to track the progress of various project components proved instrumental in ensuring that the development stayed on schedule. Platforms such as Notion and TeamGantt facilitated a transparent and organized workflow, allowing us to easily identify bottlenecks, allocate resources efficiently, and make informed decisions to keep the project on track.

## **Navigating Challenges: Dealing with Team Dynamics**

One of the notable challenges faced during the project was the presence of a team member struggling to meet expectations, jeopardizing the overall progress, and risking late submissions. This experience taught us the importance of early identification and intervention when team members face difficulties. Open communication, support, and, when necessary, reassignment of tasks was crucial in maintaining project momentum and preventing undue delays.

## **Balancing Independence and Collaboration: Individual Contributions with Collective Responsibility**

An essential lesson learned from this project was the ability to balance individual contributions with collective responsibility. While we embraced the importance of working independently on assigned tasks, we also understood the necessity of maintaining a cooperative mindset. This involved not only completing personal responsibilities but actively engaging in the broader aspects of the project. Each team member took ownership of their work, contributing to the overall success of the project by staying vigilant and ready to address challenges collectively.

In conclusion, this project provided us with a holistic understanding of project management, emphasizing the significance of effective monitoring, teamwork dynamics, and maintaining a balance between individual and collective efforts. These lessons learned will undoubtedly contribute to our future endeavours, enabling us to navigate challenges with resilience and efficiency.

# **CHAPTER VIII: CONCLUSION**

## **Summary of the Outcomes:**

### ***1.1 Eye-Catching GUI:***

* The project successfully delivered an aesthetically pleasing and engaging graphical user interface (GUI).
* Utilized modern design principles and visual elements to enhance the overall user experience.

### ***1.2 Smoothly Run Game Logics:***

* Implemented robust and efficient game logic algorithms to ensure seamless gameplay.
* Comprehensive testing and optimization resulted in a smooth and responsive gaming experience.

### ***1.3 High-Quality Images:***

* Incorporated high-resolution and visually appealing images to enhance the overall visual quality of the game.
* Leveraged optimized image assets to maintain a balance between quality and performance.

### ***1.4 Enjoyable Sounds and Videos:***

* Integrated captivating sounds and videos that complemented the gameplay and enriched the user experience.
* Ensured a harmonious blend of audio and visual elements for an immersive gaming atmosphere.

### ***1.5 Easy to Maintain Source Code:***

* Emphasized clean and modular coding practices, contributing to a codebase that is easy to understand and maintain.
* Adopted best practices for documentation, making it straightforward for future developers to grasp the project structure.

These outcomes collectively reflect the project's commitment to delivering a visually appealing, functionally robust, and enjoyable gaming experience. The focus on quality in GUI design, game logics, multimedia elements, and maintainable source code positions the project for both current success and future scalability.

## **Closing Remarks:**

In the journey of bringing this project to fruition, we have witnessed the fusion of creativity, collaboration, and technical prowess. The culmination of our collective efforts has resulted in a product that stands as a testament to our dedication to excellence. From crafting an eye-catching GUI to ensuring the smooth operation of game logic, and incorporating high-quality multimedia elements, every aspect reflects our commitment to delivering a remarkable user experience.

As we conclude this project, we take with us not only the tangible outcomes but also the invaluable lessons learned, the growth in our individual skills, and the strength derived from collaborative endeavours. This endeavour has not just been a development process; it has been a journey of discovery, resilience, and shared accomplishment.

As we look towards the future, let the successes and challenges of this project serve as stepping stones for even greater achievements. The knowledge gained, the bonds forged, and the skills honed during this project will undoubtedly shape our paths in future endeavours. This closing chapter marks not just the end of a project but the beginning of new possibilities, and we stand ready to embark on the next adventure with enthusiasm, wisdom, and a collective spirit of innovation.

# **CHAPTER IX: APPENDICES**

## **Glossary of Terms:**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| IDE | Integrated Development Environment, a software application that provides a comprehensive set of tools and features to assist developers in writing, testing, and debugging code. |
| VSC | Visual Studio Code, a lightweight, open-source code editor developed by Microsoft. It supports multiple programming languages and features extensions for enhanced functionality, making it a versatile and widely used tool for software development. |
| AI | Artificial Intelligence, it refers to the capability of machines or software to imitate human intelligence. It involves the development of algorithms and models that enable computers to perform tasks such as learning from data, recognizing patterns, and making decisions, without explicit programming. |
| GUI | Graphical User Interface, a visual interface that enables users to interact with software or electronic devices through graphical elements like icons, buttons, and windows. |
| ERD | Entity-Relationship Diagram, a visual representation used in database design to illustrate the relationships between entities. |
| UML | Unified Modelling Language, a standardized modelling language used in software engineering to visually represent the structure and behaviour of a system. |

Table 8: Glossary of Terms

## **GitHub Related Insights:**

A screenshot of a graph

Description automatically generated

Figure : Contribution to branch main (commits)

# **A graph with green and red lines Description automatically generated**

Figure : Code Frequency

# **REFERENCES**

1. YouTube Channel - Peter Mitchell:

Peter Mitchell. (n.d.). YouTube. Retrieved December 05, 2023, from https://www.youtube.com/@petermitchell7687

1. YouTube Channel - CS Students:

CS Students. (n.d.). YouTube. Retrieved December 012, 2023, from https://www.youtube.com/@csstudents1006

1. GitHub - Battleship:

Github. (n.d.). Battleship. Squirrelbear. Retrieved December 05, 2023, from https://github.com/Squirrelbear/Battleship