

War-Themed game

────

Guess the location

────

Attractive fighting

────

Play with enemy

OCTOPUS BATTLE

GAME DEVELOPER

**Tran Tuan Nghiep**

**Nguyen Thien Bao**

**Le Cong Thai Khang**

**Đinh Binh Thanh Thong**

**Dang Khai Duong**

ADVISOR

Prof. Tran Thanh Tung

# FINAL PROJECT DSA

*HCMIU*

The Big Welcome to the deep sea. Where big creatures appear and this is the battle between them (octopus). Focus on the map and try to set up your formation in the most unpredictable way possible (the enemy will do the same to you). Then rush into battle and wipe out all enemies in your way by guess the hidden location of their octopus army. The war only ends when one of the two sides is wiped out.

Contents

[CHAP 1: INTRODUCTION 5](#_bookmark0)

* 1. [Background: 5](#_bookmark1)
  2. [Game project 5](#_bookmark2)
  3. [Game rules: 6](#_bookmark3)
  4. [References: 6](#_bookmark4)
  5. [Developer team: 7,8](#_bookmark5)

[CHAP 2: SOFTWARE REQUIREMENTS 9](#_bookmark6)

* 1. [What we have: 9](#_bookmark7)
  2. [What we want: 9](#_bookmark8)
  3. [Working tools, platform 10](#_bookmark9)
  4. [Use Case Scenario 10](#_bookmark10)
  5. [Use Case diagram 11](#_bookmark11)
  6. [Class diagram 12](#_bookmark12)

[CHAP 3: DESIGN & IMPLEMENTATION 13](#_bookmark17)

[Package Diagram 13](#_bookmark18)

[UI Design 13](#_bookmark19)

[Main and frame 13](#_bookmark20)

* + 1. [Compute 14](#_bookmark21)
    2. [Battle frame: 18](#_bookmark22)
    3. [Manage Octopus Frames: 19](#_bookmark23)
    4. [Splash Screen Frame 19](#_bookmark24)

* + 1. [Map 20](#_bookmark25)
    2. [Octopus Positions 20](#_bookmark26)
    3. [Positions 21](#_bookmark27)
    4. [Report 22](#_bookmark28)
    5. [BackGround 23](#_bookmark30)
    6. [Manage Panel: 24](#_bookmark31)
    7. [Map Panel 24](#_bookmark32)
    8. [Stat Panel 24](#_bookmark33)

[CHAP 4: FINAL APP GAME 25](#_bookmark40)

[Source code (link github) 25](#_bookmark41)

1. [Instruction: 25](#_bookmark44)
2. [Main Menu: 25](#_bookmark45)
3. [Step 1: 25](#_bookmark46)
4. [Step 2 25](#_bookmark47)
5. [Step 3: 26](#_bookmark48)

[CHAP 5: EXPERIENCE 27](#_bookmark51)

# CHAP 1: INTRODUCTION

## Background:

In the context of designing and creating a game in preparation for the final DSA course presentation this semester. Our team of 5 members decided and agreed to create a real-time war themed game based on player-computer interaction. In particular, this game must meet the content and knowledge taught in class in this DSA course (algorithms, object-oriented programming, etc.).

The game requires graphics that are easy to see and friendly but not too simple. And the game must also have in-depth mechanisms and good interaction with users

## Game project:

With the above mentioned theme of interaction based on player actions.

We decided to create a war game in the middle of the ocean. There is a war between big and small octopuses fighting each other aiming to dominate and take over the enemy's territory.

Imagine that you will play the role of the orchestrator of that fight (positioning our octopuses). As well as then attack by guessing the enemy's hiding location (the positions the enemy has established). Since both players are already in position and do not know where the other is, the game requires a reasonable combination of guessing and discarding during their turns as logically and optimally as possible.

## Game rules:

As mentioned above. Game rules include 2 main parts: setup and combat. In the setup turn, the player will position his 4 types of octopus in the desired direction (horizontal or vertical) on the available map. If you are not satisfied and place it incorrectly, the player can press RESET (which causes the map to return to the original) or if you want to start immediately, you can try the RANDOM button (randomly placing the octopus positions).Add more sounds and animation to make the game more attractive.

Through the main part (battle part), players will go to 2 interfaces (1 for themselves and 1 for the opponent). Look on the enemy map and guess the location of their octopus army by clicking on the box on it. If there is nothing, the box simply displays an empty effect, but if it hits the opponent, it will display a red icon. The opponent can also do the same thing to you and if they choose the correct position of hidden troops, a death sign will appear on the player's map. The turns will rotate until one of the two sides is completely wiped out on the board.

## References:

* [You Sank My Battleship! A Case Study in Secure Programming (mit.edu)](https://www.ll.mit.edu/sites/default/files/publication/doc/2018-04/2014_07_29_ZhivichM_PLAS_FP.pdf?fbclid=IwAR2yiDSJeUDGXlkk-Re057tfznsl3B0sftTgTjSEoQxuct8iq_QGm4EwT-Y)[Battleship (game) - Wikipedia](https://en.wikipedia.org/wiki/Battleship_(game)?fbclid=IwAR2Qz_FD1Yb6H6I3H3Fz2BqcM00H3QPrfnQWN8xMPvlBmcmqoqEjA8Zu-XY)
* [RPubs - In A Game of Battleship, Strategy Influences the Outcome Given You Are Doing It Right](https://www.rpubs.com/cbennage/battleship_optimal_strategy_research?fbclid=IwAR0jEHhccNjIZlpPB3XGjb6YDo0Pt1sOrz-ihmLzDUwRKOd-tpUbReDitjk)\
* [Battleship (game) - Wikipedia](https://en.wikipedia.org/wiki/Battleship_(game)?fbclid=IwAR2Qz_FD1Yb6H6I3H3Fz2BqcM00H3QPrfnQWN8xMPvlBmcmqoqEjA8Zu-XY)

## Developer team:

Our of us is a Data Structure Algorithm project team. We have 5 members come from International University, Computer Science major:

|  |  |  |
| --- | --- | --- |
| **Name**  **- Github username** | **UID** | **Contribute** |
| Tran Tuan Nghiep | ITITIU20259 | Leader of team  Write report  Design game |
| Nguyen Thien Bao | ITITIU19022 | Write report  Design Motion event  Support main function of handle game |
| Le Cong Thai Khang | ITITIU20224 | Draw diagrams  Design UI and effect  Support functions of algorithms |
| Dinh Binh Thanh Thong | ITITWE19027 | Write slides  Support every part of the project Fix bugs and test game |
| Dang Khai Duong | ITITWE19010 | Write slides  Support every part of the project Fix bugs  Test game |

# **CHAP 2: SOFTWARE REQUIREMENTS**

## What we have:

1. User friendly efficient and lucrative system.
2. Minimum maintenance cost (graphics).
3. A highly competitive strategy game
4. Easy to operate, easy to run on many programming environments
5. Suitable for entertainment and replay

## What we want:

1. Develop system within limited cost.
2. Friendly, easy-to-understand interface
3. Design the game mechanics are deep and antagonistic
4. Provides a real interactive opponent
5. Easy to update.

## Working tools, platform: ( bổ sung):

## Built on the java language.

## Jpanel container

## Package, library:

## java.io.

## java.awt.event.

## javax.swing

## java.util.

## java.imageio

## Use Case Scenario:

We have created the use cases based on the UX view of the game.

|  |  |  |
| --- | --- | --- |
| **OCTOPUS BATTLE** | SET UP | Choose the octopus (4 types) |
| Place them in verticles or horizontal |
| RANDOM ( if player want to random set up) |
| RESET | Reset all (return to the initial screen) |
| FIGHT | Start a game |
| BACK | Exit game ( yes) or continue (no) |

## Use Case diagram:

A diagram of a computer

Description automatically generated

## Class diagram:

## Không có mô tả.

**CHAP 3: DESIGN & IMPLEMENTATION**

**Package Diagram**

**UI Design**

**Resources files will be stored in res/images folder.**

All these files are png and gif, and they are assets for main program

A screenshot of a computer

Description automatically generated

**Main method**

The code represents the main class of a simple program that creates and displays two graphical user

**interface (GUI)** frames. A screen shot of a computer code

Description automatically generated

An instance of the FrameSplashscreen class is created and assigned to the variable intro, with the purpose of displaying a splash screen.

A computer code with text

Description automatically generated with medium confidence

This block of code attempts to make the main thread sleep for 2000 milliseconds (2 seconds). This is done to introduce a delay, possibly to display the splash screen for a short period before moving on to the next frame. The Thread.sleep method can throw an InterruptedException, but the code ignores it by having an empty catch block.



After the delay, the visibility of the splash screen frame (intro) is set to false, effectively hiding it.



Then a new instance of the FrameManageOctopus class is created and assigned to the variable manage.UI

1. **Computer**

This class is named Computer, which represents a simplified computer player that attacks the opponent's map strategically by making hits and handling the consequences of hits, such as sinking octopus-shaped objects and adjusting hit directions.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screen shot of a computer code

Description automatically generated

This method is responsible for determining the computer player's action in a turn-based game involving an octopus-themed map. It determines the computer's actions based on the number of hits made, utilizing methods like hitRandomly, wrongHit1, and wrongHit2 for different hit scenarios. The method returns a Report object containing information about the computer player's turn, such as the hit position, hit status, and whether an octopus was sunk.

A computer screen shot of a program code

Description automatically generated

This method is responsible for making a random hit on the opponent's map. Here's how it works:

* It generates a random index (attackNo) within the range of available hits (listOfHits.size()).
* It removes the Position object at the generated index from listOfHits.
* The removed Position is stored as the lastHit.
* It performs a hit (plMap.hitt(p)) on the selected position.
* It returns a boolean indicating whether the hit was successful.

A computer code with text

Description automatically generated

This method represents the strategy for the computer player when it has made the first hit. It attempts to make a hit based on the first hit's location and direction. The steps are as follows:

* It generates a random index (tiro) within the range of available directions (possibility.size()).
* It removes the selected direction from possibility.
* It creates a new Position object (p) based on the first hit's location and moves it in the selected direction.
* It updates the direction with the selected direction.
* If the selected position is not water (plMap.acqua(p)), it removes it from the available hits (listOfHits).
* It repeats these steps until a valid position is selected (not hitting an already hit position).
* The selected position (p) becomes the lastHit.
* It performs a hit (plMap.hitt(p)) on the selected position.
* It returns a boolean indicating whether the hit was successful.

A screen shot of a computer code

Description automatically generatedThis method represents the strategy for the computer player when it has made two or more hits. It continues hitting in a specific direction determined by the direction field. The steps are as follows:

* It creates a new Position object (p) based on the last hit's location.
* It moves the position in the stored direction.
* If the position is out of the map or hits water (plMap.acqua(p)), it calls revertDirection() to change the direction.
* If the position is not water and not already hit (!plMap.hit(p)), it sets isHit to true, indicating a successful hit.
* It repeats these steps until a successful hit is made.
* The selected position (p) is removed from the available hits (listOfHits).
* The lastHit is updated with the selected position.
* It performs a hit (plMap.hitt(p)) on the selected position.
* It returns a boolean indicating whether the hit was successful.

The method removeOutlines(OctPos dead) is responsible for removing the possible hit positions around a sunk octopus from the list of available hits (listOfHits). It also A screenshot of a computer code

Description automatically generatedmarks those positions as water in the player's map (plMap).

The initializeHit method is part of the computer player's strategy in the game. It is responsible for populating the possibility linked list with directions for potential hits based on the last successful hit's position. The method takes into account the boundaries of the game map (Map.DIM\_MAP) to determine the valid directions for further hits.

The revertDirection method is responsible for changing the current hit direction of the computer player to the opposite direction. This method is used when the computer player encounters a situation where it cannot continue hitting in the current direction, such as hitting the boundary of the game map or water.A computer code with text

Description automatically generated

1. **Battle Frame**

This code defines a class named FrameBattle, which is a part of our battleship game.

Methods:

* setAttack: Sets the attack result on the GUI based on the provided Report.
* actionPerformed: Handles actions when a button is clicked.
* setDeadSquid: Handles the GUI display when a squid is defeated.
* deleteSquid: Updates the statistics panel when a squid is defeated.
* keyPressed: Handles key presses, including an escape key for returning to the main menu.
* TimeController: An ActionListener for handling CPU turns.
* drawTarget: Draws the target on the game board.
* handleSaveClick: Handles saving the game state.
* drawLoad: Draws the game state after loading.
* load: Loads the game state from saved files.
* handleBackClick: Handles returning to the main menu.

1. **Manage Octopus Frame**

This class FrameManageOctopus is another part of our battleship game. It's responsible for managing the placement of octopuses on the player's game board before entering into the battle phase.

Action Listener Implementation (actionPerformed method):

* Handles button clicks for reset, random, and fight actions.
* Handles octopus placement when a specific cell on the game board is clicked.
* Updates counters, UI elements, and the game board accordingly.

Key Listener Implementation (keyPressed method):

* Allows certain actions to be triggered by keyboard input.
* 'G': Randomly places octopuses and initiates the battle.
* 'R': Randomly places octopuses.
* DELETE/BACKSPACE: Resets the game state.
* ESCAPE: Exits the game.
* ENTER: Initiates the battle if octopus placement is complete.

Methods:

* random: Places octopuses randomly on the board.
* reset: Resets the game state, allowing the player to reposition octopuses.
* fight: Initiates the battle phase by creating an instance of FrameBattle.

1. **Splash Screen Frame**

The FrameSplashscreen class represents a splash screen for a Battleship game. Here's an overview of its structure and functionality:

* initializeUI() Method:
  + Configures the JFrame to be undecorated (no window borders) and non-resizable.
  + Sets the size and icon of the splash screen.
  + Creates a background panel (UIJPanelBG) with an image loaded from the resources.
  + Creates a JLabel with a loading image (loading.gif) and adds it to the splash screen.
  + Makes the splash screen visible.
  + Simulates a loading process by using Thread.sleep(3000) (3 seconds). Adjust the sleep time as needed.
  + Disposes of the splash screen when the loading is complete.

* centerFrameOnScreen() Method:
  + Centers the splash screen on the screen by calculating the appropriate X and Y coordinates.

1. **Map**

This class Map represents a game map for a battleship-like game where octopuses are placed on the map. The class has methods to initialize and manipulate the map, as well as check for hits and sunk octopuses.

* initializeRandomMap() Method:
  + Clears the map and randomly inserts octopuses of different dimensions at random positions.
* clear() Method:
  + Resets the map by setting all elements to the NULL state.
* insert(int x, int y, int dim, int dir) Method:
  + Inserts an octopus on the map at the specified position, dimension, and direction (horizontal or vertical).
* insertRandom(Random random, int dimension) Method:
  + Inserts a random octopus of a specified dimension at a random position and direction.
* checkVertical(int row, int column, int dimension) and checkHorizontal(int row, int column, int dimension) Methods:
  + Check if it's valid to insert an octopus vertically or horizontally at the specified position.
* hitt(Position p) Method:
  + Checks if a position on the map contains an octopus and updates the map accordingly.
* sunk(Position p) Method:
* Checks if an octopus at a specific position has been sunk (all parts hit), and if so, removes it from the list.
* setAcqua(Position p), acqua(Position p), and hit(Position p) Methods:
* Set and check the state of a position as water or hit, respectively.
* toString() Method:
* Generates a string representation of the map for debugging and display purposes.
* setAdvOctopus(LinkedList<int[]> advOctopus) Method:
* Sets the octopus positions on the map based on the provided list of integer arrays.

1. **Octopus Positions**

The OctPos class represents the position of an octopus on the game map and provides methods for accessing, checking, and comparing octopus positions.

A screenshot of a computer code

Description automatically generated

1. **Positions**

The Position class is designed to represent a position on a map, allowing for movement in different directions and providing methods for comparing and checking the position.

A screenshot of a computer code

Description automatically generated

1. **Report**

The Report class is designed to encapsulate information about an attack, providing details about the attack's coordinates, whether it hit, and whether it resulted in the sinking of an octopus.

A screenshot of a computer code

Description automatically generated

1. **Background**

The UIJPanelBG class is designed to be a customized JPanel with a background image. The background image can be specified either through a file path or by providing an Image object directly. The paintComponent method ensures that the background image is drawn when the panel is painted. The createImageIcon method is a utility method for creating an ImageIcon from a file path.

A screenshot of a computer code

Description automatically generated

1. **Manage Panel**

The UIManagePanel class is designed to provide a graphical user interface for managing octopuses in a game scenario. It extends UIJPanelBG, which provides a background image for the panel. The UI elements include options for selecting octopuses, specifying directions, and triggering various actions. The layout and appearance are configured to create an interactive and visually appealing user interface.

A screenshot of a computer code

Description automatically generated

1. **Map Panel**

The UIMapPanel class represents a panel displaying a grid of buttons, each representing a cell in the game map. The panel includes buttons for the player's map and labels for coordinates. It includes functionality to draw octopuses on the grid based on specified parameters. The layout and appearance are configured to create an interactive and visually appealing user interface.

A screenshot of a computer program

Description automatically generated

1. **Stat Panel**

The UIStatPanel class is designed to provide a visual representation of octopus statistics during a battle. The layout and appearance are configured to display octopus images in a specific arrangement on the panel. The statistics may include information such as the type and count of octopuses involved in the battle.A screen shot of a computer code

Description automatically generated

**CHAP 4: FINAL APP GAME**

Source code (link github): <https://github.com/TuanNghiep/DSABattleship2023>

Instruction:

* Step 1: Place all octopus on the ocean grid, each octopus can be placed horizontally or vertically across grid spaces. You can use the “Random” button to automatically place octopus on the grid, or “Reset” button to reset all positions.

A screenshot of a game

Description automatically generated

* Step 2: Press “Fight” once you are done with placement. You cannot change the position of the octopuses after the game begins.
* Step 3: Player takes first turn firing shots (by calling out a grid coordinate) to attempt to hit the opponent's enemy octopuses. The game will call out if the shot is hit or miss. When you miss, the grid becomes empty and it's the computer's turn firing its shot. When you hit, the game marks that grid with an image of the body that got hit and gives you an extra turn each time you hit, until you hit all parts of the octopus, the game marks that octopus as destroyed.

A screenshot of a game

Description automatically generated

# **CHAP 5: EXPERIENCE:**

Throughout the process of collaborating and perfecting the game until the completion and completion of the project, we have left us with a lot of valuable experience.

Thanks to game production, we have gained more experience in applying JPanel, Java, IntelliJ IDEA platform, etc.. in our group's own projects.

The project applied the knowledge learned in the DSA course and previous courses in programming and applying algorithms and data structures. It is possible that in this release project there will be outstanding issues and unsuitability for some users, but we will update and upgrade regularly.

If we can judge for ourselves, we see that the project basically meets the pre-set requirements and is also qualified to satisfy the requirements of this course. However, on the downside, it can be said that we can improve the game better in some ways (multiplayer mode, dynamic animation, etc.) so the team will hone and improve on these projects later sentence.

In conclusion, thank you for taking the time to read, follow our project and support. Hopefully the project will bring benefits to consumers.

- END -