INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY, HCM CITY

School of Computer Science & Engineering

FINAL REPORT

<u>TOPIC: SPACE-INVADER-MASTER</u>

<u>Lecturer</u>: <u>*Dr. Tran Thanh Tung*</u>

<u>Course:</u> Object-oriented-Programming 2023-2024

Group 4's member:

Nguyễn Hoàng Việt / Phạm Nhật Tân /Ngô Quang Hải

Nguyễn Đức Quốc Anh / Đào Trọng Khoa

TABLE OF CONTENTS

CONTRIBUTION TABLE

GROUP 4			
Name	ID	Task	Contribution
Nguyễn Hoàng Việt	ITITIU20351	Alien, won, shot, report	27,5%
Phạm Nhật Tân	ITITUN19007	common, board, gameover, report	15%
Ngô Quang Hải	ITCSIU22276	Spaceinvader, Presentation slide	27,5%
Nguyễn Đức Quốc Anh	ITITWE20018	Player, Push Panel, Presentation Slide	15%
Đào Trọng Khoa	ITITIU19145	Sprite, Bomb, UML Design	15%

ABSTRACT

The Space Invader game, an iconic classic in the realm of arcade gaming, has been reimagined and brought to life in the Java programming language. Originating from the late 1970s, Space Invaders has transcended generations, leaving an indelible mark on the gaming community. This project serves as an exploration into the intersection of creativity and code, as it recreates the immersive experience of defending Earth against descending alien invaders.
This report encapsulates the journey of developing the Space Invader game in Java, highlighting key aspects of the implementation process. Leveraging the object-oriented features of Java, the project focuses on graphics rendering, user input handling, and game logic to recreate the classic gameplay. Additionally, the report delves into the challenges faced during development and the application of fundamental programming concepts. Through this endeavor, the aim is not only to resurrect the nostalgia of Space Invaders but also to provide insights into the practical application of Java programming in game development. The project incorporates Object-Oriented Programming (OOP) concepts such as encapsulation, inheritance, and polymorphism, showcasing their relevance and effectiveness in creating a cohesive and engaging gaming experience.

As we navigate through the intricacies of code and design, the Space Invader game in Java becomes a canvas for learning and innovation. The abstract nature of space and the structured world of programming converge in this project, demonstrating the dynamic synergy between creativity and technology in the realm of game development.

Chapter 1: INTRODUCTION

A. Objectives

In the ever-evolving realm of computer programming, game development stands out as a captivating and challenging domain. The fusion of creativity and technical prowess brings virtual worlds to life, captivating audiences of all ages. Among the plethora of games that have graced the digital landscape, Space Invaders holds a timeless charm, etching its place as a classic arcade game since its inception in the late 1970s.

This report delves into the creation and exploration of a Java implementation of the iconic Space Invader game. Leveraging the object-oriented capabilities of Java, this endeavor involves crafting a virtual space battleground where players navigate a spaceship, battling hordes of descending alien invaders. The project encapsulates fundamental programming concepts, including graphics rendering, user input handling, and game logic implementation. The significance of this undertaking extends beyond the mere recreation of a nostalgic gaming experience. It serves as an educational exploration into the intricacies of Java programming, allowing developers to hone their skills while building a tangible and entertaining application. The report outlines the structural design, key functionalities, and challenges encountered during the development process, shedding light on the inner workings of the Space Invader game in Java.

As we embark on this journey through code and creativity, the Space Invader game in Java becomes not only a testament to the enduring legacy of classic gaming but also a canvas upon which programming skills are refined and elevated. Let the pixels illuminate the screen as we delve into the intergalactic world of Space Invaders in the Java programming language.

To be short, the project aims to:

- Create a game that is redesigned to entertain and educate players.
- Practicing OOP techniques in the Theory class.
- Go through the process of game management and code refining.
- Evaluate the ability to build more features on top of the basic code.

2. The tools used

- IDE for programming and debugging:Visual studio code.
- Mean of code version management: GitHub.
- Means of contacting: Notion and Microsoft Teams.

CHAPTER 2: METHODOLOGY

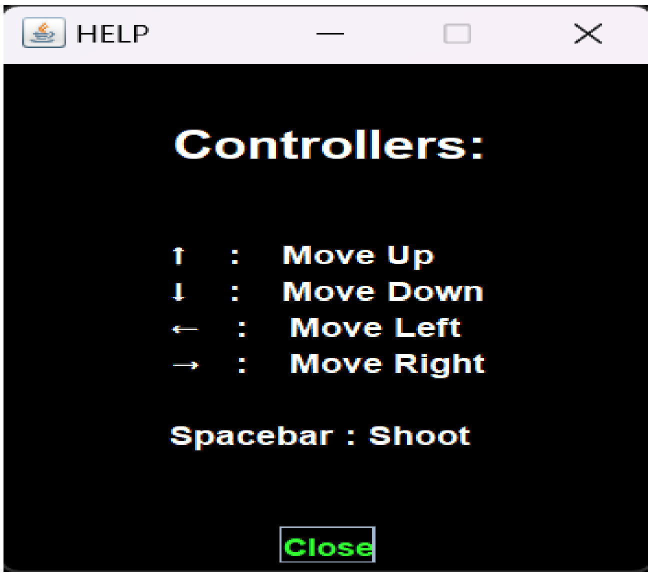
1.Rules

Objective:

Defend the planet against invading alien forces by shooting down alien spaceships.

Player Controls:

Use the arrow keys to move the player's spaceship:
← (Left Arrow): Move Left
→ (Right Arrow): Move Right
↑ (Up Arrow): Move Up
↓ (Down Arrow): Move Down
Use the spacebar to shoot projectiles.


HELP

Controllers:

↑ : Move Up
↓ : Move Down
← : Move Left
→ : Move Right

Spacebar : Shoot

Alien Invasion:

Aliens will descend from the top of the screen in formation.
The player must prevent the aliens from reaching the bottom of the screen.

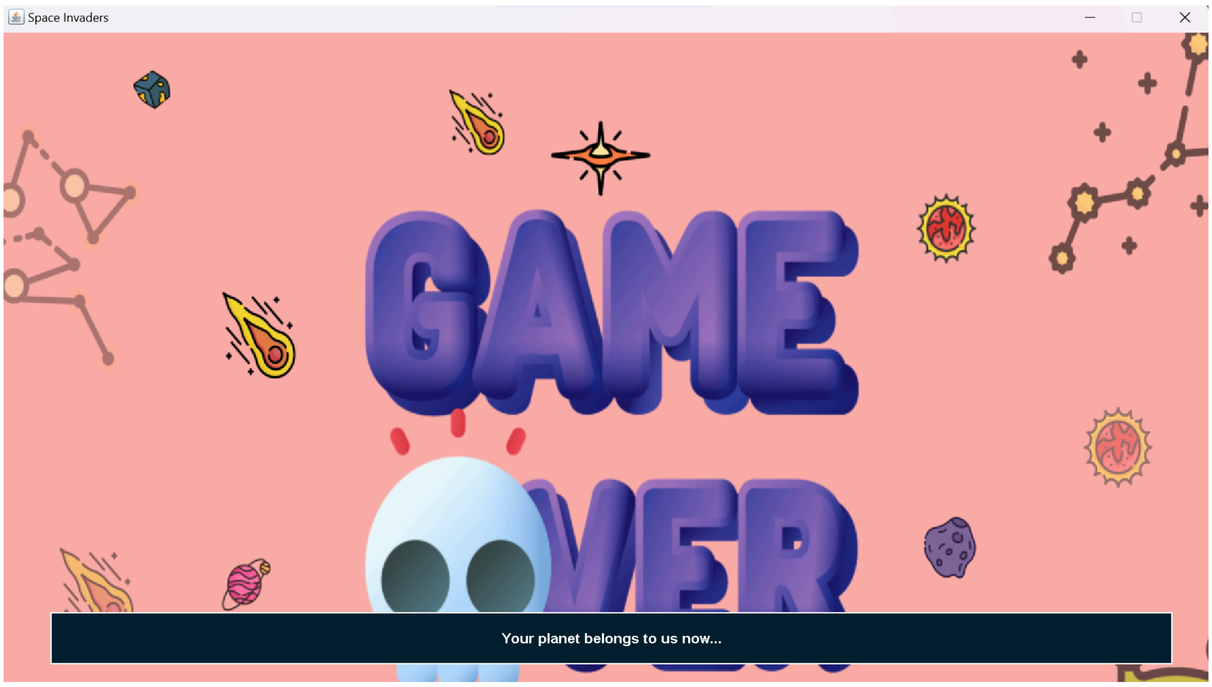
Player's Spaceship:

The player's spaceship starts at the bottom center of the screen.
The spaceship has limited movement within the game boundaries.

Alien Spaceships:

Aliens move horizontally and descend vertically as a group.
The player must eliminate aliens by shooting them with projectiles.

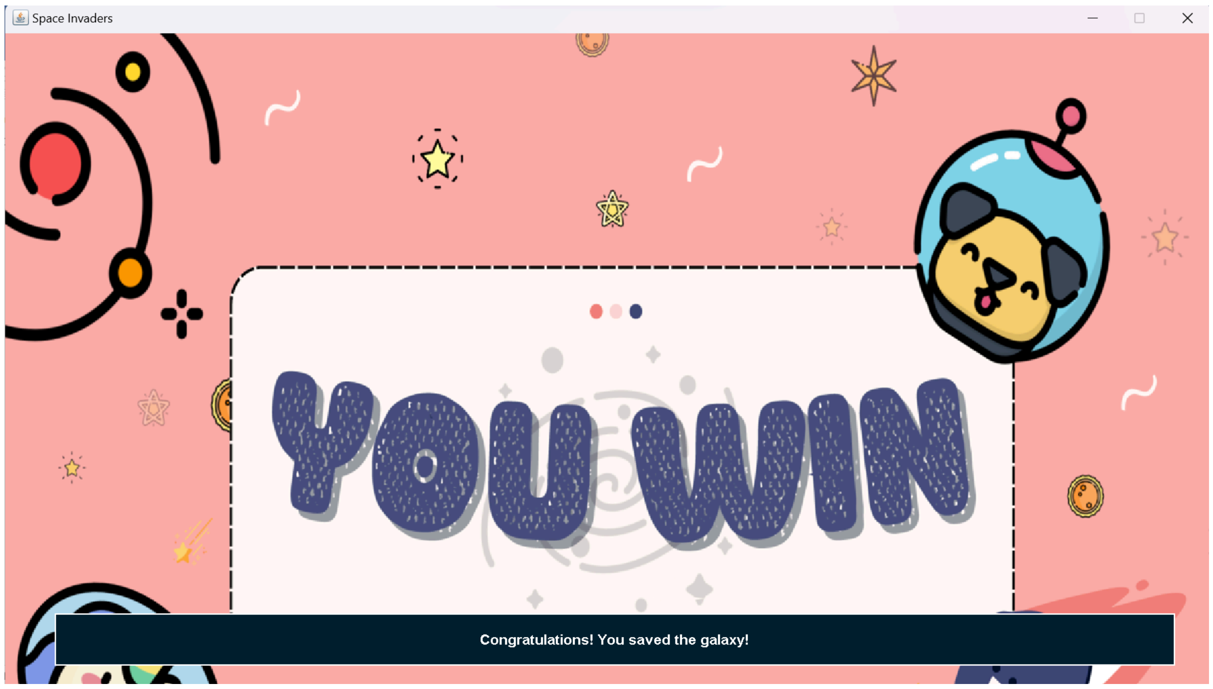
Projectiles:
The player can shoot projectiles to eliminate alien spaceships.
Projectiles travel vertically and disappear upon reaching the top of the screen or hitting an alien.
Alien Bombs:
Aliens periodically drop bombs towards the player's spaceship.
If a bomb reaches the player's spaceship, the player loses a life.
Player Lives:
The player starts the game with a set number of lives.
Losing all lives results in the end of the game.
Game Over:
The game ends if the aliens reach the bottom of the screen.
The game ends if the player's spaceship is hit by an alien bomb and has no remaining lives.



Your planet belongs to us now...

Victory:
The player wins by eliminating all the monster.

Victory results in a congratulatory message and the end of the game.



Congratulations! You saved the galaxy!

Scoring:
Players earn points for each alien spaceship eliminated.
Additional points may be awarded for completing levels or achieving specific milestones.
Difficulty Levels:
The game may include multiple difficulty levels, affecting the speed and behavior of the aliens.

Restarting the Game:
Players can restart the game after reaching a game-over state.

High Score:
The game may track and display the player's highest score achieved across multiple sessions.

Sound Effects and Visuals:
The game includes sound effects for actions such as shooting, alien destruction, and player hits.
Visual cues, such as explosions, enhance the gaming experience.

Help and Instructions:
Players can access instructions and help screens to understand the game controls and objectives.

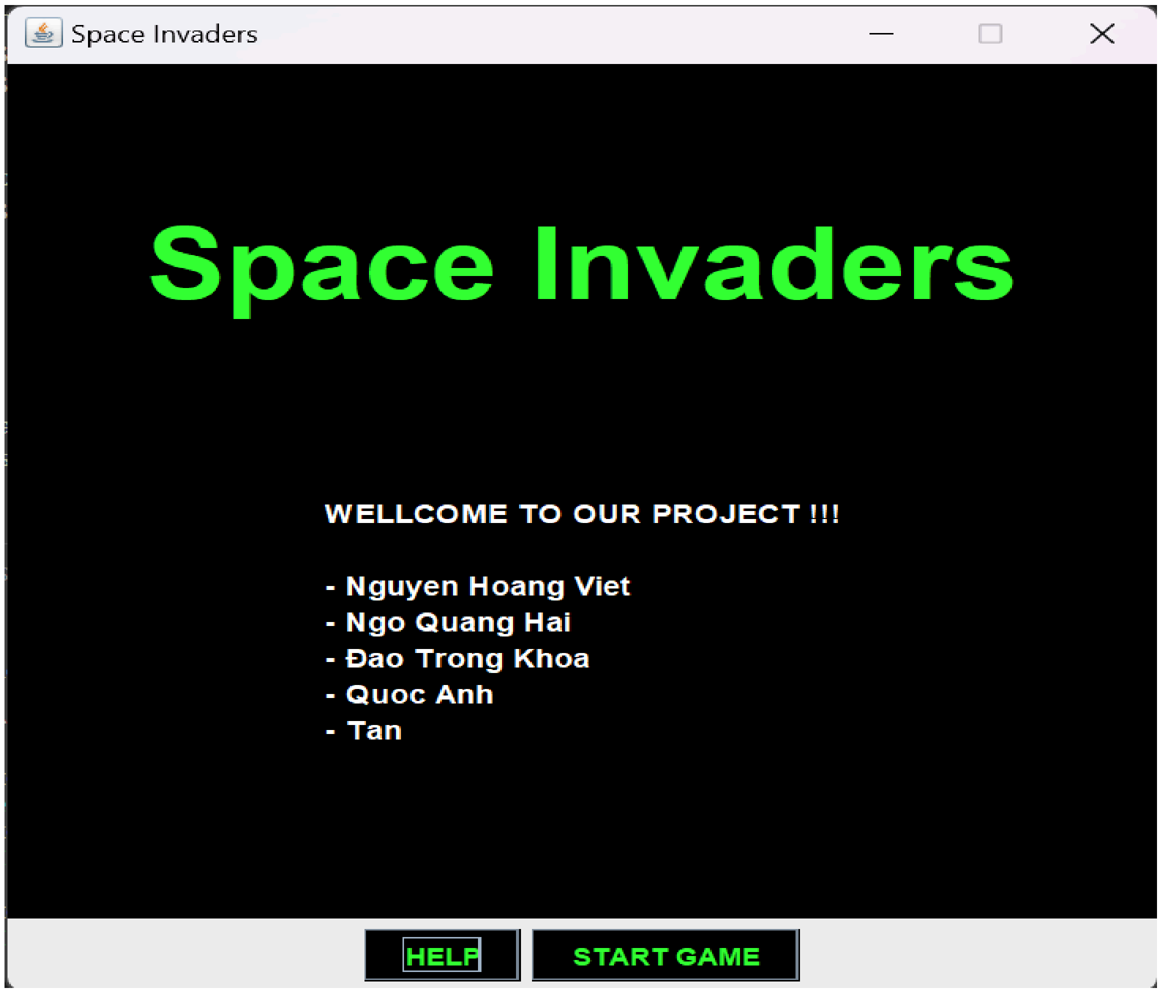
Pause and Resume:
The game may provide an option to pause and resume gameplay.

Enjoy the Game:

The primary goal is to provide an engaging and enjoyable gaming experience for players.

2. Design

Visual Style:
The game adopts a retro-inspired, pixel-art visual style reminiscent of classic arcade games.
Vibrant colors and distinct design elements contribute to an appealing and nostalgic aesthetic.



WELLCOME TO OUR PROJECT !!!

- Nguyen Hoang Viet
- Ngo Quang Hai
- Đao Trong Khoa
- Quoc Anh
- Tan

User Interface (UI):
The UI features essential components such as player score, remaining lives, and level indicators.
Clear and intuitive on-screen controls guide players on how to navigate the game.

Game Elements:

- Player Spaceship:

Positioned at the bottom center, the player's spaceship is designed with attention to detail, offering a futuristic yet accessible appearance.

- Alien Spaceships:

Aliens form a formation at the top of the screen, showcasing a variety of unique designs for visual diversity.
Animations depict alien movements and reactions upon being hit.

- Projectiles:

Player projectiles and alien bombs have distinct appearances and animations, making them easily distinguishable during gameplay.

Animation and Special Effects:
Smooth animations enhance the overall gaming experience.
Explosions, visual cues for successful hits, and dynamic movement of game elements contribute to the game's excitement.
Background and Setting:
The game is set against a space-themed background featuring stars, planets, and other celestial elements.
Background visuals may change or evolve as players progress through levels, adding variety.

Sound and Music:
An upbeat and dynamic soundtrack complements the fast-paced gameplay, contributing to the overall atmosphere.
Sound effects for shooting, alien destruction, and other actions enhance player engagement.

Level Design:
The game includes multiple levels, each progressively challenging.
Level backgrounds and alien formations change, providing a sense of progression and variety.

Game Logic and Mechanics:
The core game mechanics follow classic arcade principles with a focus on simplicity and accessibility.
Alien movements, projectile trajectories, and collision detection are designed for balanced and enjoyable gameplay.

Challenges and Rewards:
The game introduces increasing challenges as players advance through levels.
Rewarding players with points, visual effects, or power-ups for achieving specific milestones adds motivation.

User Experience (UX):
Intuitive controls and a well-designed HUD contribute to a user-friendly experience.
Clear feedback mechanisms, such as scoring animations and visual indicators for remaining lives, enhance user understanding.

Replayability:

The game is designed for replayability, encouraging players to return for improved scores and conquering higher levels.
Randomized elements or difficulty adjustments maintain excitement during repeated plays.

Accessibility:
Consideration is given to make the game accessible to players with different skill levels.
Adjustable difficulty settings or tutorial levels cater to both casual and experienced gamers.

Testing and Iteration:
Extensive playtesting is conducted to identify and address any issues related to game balance, controls, and overall enjoyment.
Feedback from playtests informs iterative improvements to enhance the overall design.

Game algorithm:

Alien Class:

- □ **act(int direction):**
- Purpose: Update the position of the alien based on the given direction.
- Input: direction - An integer indicating the direction of movement.
- Logic:Update the x coordinate of the alien based on the direction parameter.
- □ **getBomb():**
- Purpose: Get the bomb associated with the alien.
- Output: Returns the bomb object associated with the alien.

Board Class:

- □ **gameInit():**
- Purpose: Initialize the game by creating alien objects, player object, and shot object.

  Logic:
- Create a list of alien objects and position them in a grid.
- Create a player object.
- Create a shot object.
- Start the game loop.

- □ **drawAliens(Graphics g):**
- Purpose: Draw the alien objects on the game board.
- Input: g - Graphics object for drawing.
- Logic:
  Iterate through the list of aliens.
  If an alien is visible, draw its image at its current position.
  If an alien is dying, update its state.

 drawPlayer(Graphics g):
- Purpose: Draw the player object on the game board.
- Input: g - Graphics object for drawing.
- Logic:
  If the player is visible, draw its image at its current position.
  If the player is dying, update its state and set havewon and ingame accordingly.

 drawGameEnd(Graphics g):
- Purpose: Draw the game-over or victory screen.
- Input: g - Graphics object for drawing.
- Logic:
  Draw the appropriate image based on whether the player won or lost.

 drawShot(Graphics g):
- Purpose: Draw the shot on the game board.
- Input: g - Graphics object for drawing.
- Logic:
  If the shot is visible, draw its image at its current position.

 drawBombing(Graphics g):
- Purpose: Draw the bombs dropped by aliens on the game board.
- Input: g - Graphics object for drawing.
- Logic:
  Iterate through the list of aliens.
  If an alien's bomb is not destroyed, draw its image at its current position.

 paint(Graphics g):
- Purpose: Paint the entire game board, including aliens, player, shot, and bombs.
- Input: g - Graphics object for drawing.
- Logic:
  Fill the background with a black color.
  Draw aliens, player, shot, and bombs.
  Sync the graphics and dispose of the graphics object.

 gameOver():
- Purpose: Handle the game-over state.
- Logic:
  Draw the game-over or victory screen.
  Display appropriate messages based on the outcome.

 animationCycle():
- Purpose: Manage the animation cycle of the game.
- Logic:
  Check if the player has won by destroying all aliens.
  Update the player's position.

Handle shot collisions with aliens.
Move the alien fleet.
Handle bomb drops from aliens.
Update bomb positions and check for collisions with the player.
Control the game loop with time delays.

 run():
- Purpose: Run the game loop in a separate thread.
- Logic:
  Continuously repaint the game board and update the game state.
  Introduce a time delay to control the frame rate.

 addNotify():
- Purpose: Add notification when the game is ready to be displayed.
- Logic:
  Call the **‘gameInit()’** method.

Bomb Class:

 Bomb(int x, int y):
- Purpose: Constructor to initialize bomb properties.
- Input: Initial x and y coordinates.
- Logic:
  Set initial coordinates, set the image, and mark the bomb as destroyed.

“setDestroyed(boolean destroyed) and isDestroyed():”
- Purpose: Getter and setter for the bomb's destroyed state.
- Logic:
  Get or set the destroyed property.

Commons Interface:
- Purpose: Define common constants for the game, such as board dimensions, alien and player sizes, etc.

GameOver Class:

 GameOver():
- Purpose: Constructor to initialize properties of the game-over screen.
- Logic:
  Set the image and initial coordinates.

Player Class:

 act():
- Purpose: Update the player's position based on the current velocity.
- Logic:
  Update x and y coordinates based on the current velocity.
  Restrict player movement within the screen boundaries.

“keyPressed(KeyEvent e)” and “keyReleased(KeyEvent e)”

- Purpose: Handle key presses and releases for player movement.
- Input: e - KeyEvent object.
- Logic:
  Update player movement based on key events.

PushPanel Class:

- **ButtonListener (inner class):**
- Purpose: Handle button clicks to start the game.
- Logic:
  Create a new instance of the **"SpaceInvaders"** class when the button is clicked.

Shot Class:

- **Shot() and Shot(int x, int y):**
- Purpose: Constructors to initialize shot properties.
- Logic:
  Set initial coordinates and image.

SpaceInvaders Class:

- **SpaceInvaders() (Constructor):**
- Purpose: Initialize the main frame and components for the game.
- Logic:
  Set up the introductory screen with buttons for starting the game and accessing help.
  Create instances of the Board class when the game starts.

- **closeIntro() and closeHelp():**
- Purpose: Close the introductory and help screens.
- Logic:
  Dispose of the corresponding frames.

- **'ButtonListener' and 'HelpButton' (inner classes):**
- Purpose: Handle button clicks for starting the game and accessing help.
- Logic:
  Create instances of the SpaceInvaders class and help screen, respectively.

Sprite Class:

'die()', 'isVisible()', 'setVisible(boolean visible)', 'setImage(Image image)', 'getImage()', 'setX(int x)', 'setY(int y)', 'setDying(boolean dying)', 'isDying():'

- Purpose: Provide common methods and properties for game entities.
- Logic:
  Set or get visibility, image, coordinates, and dying state.

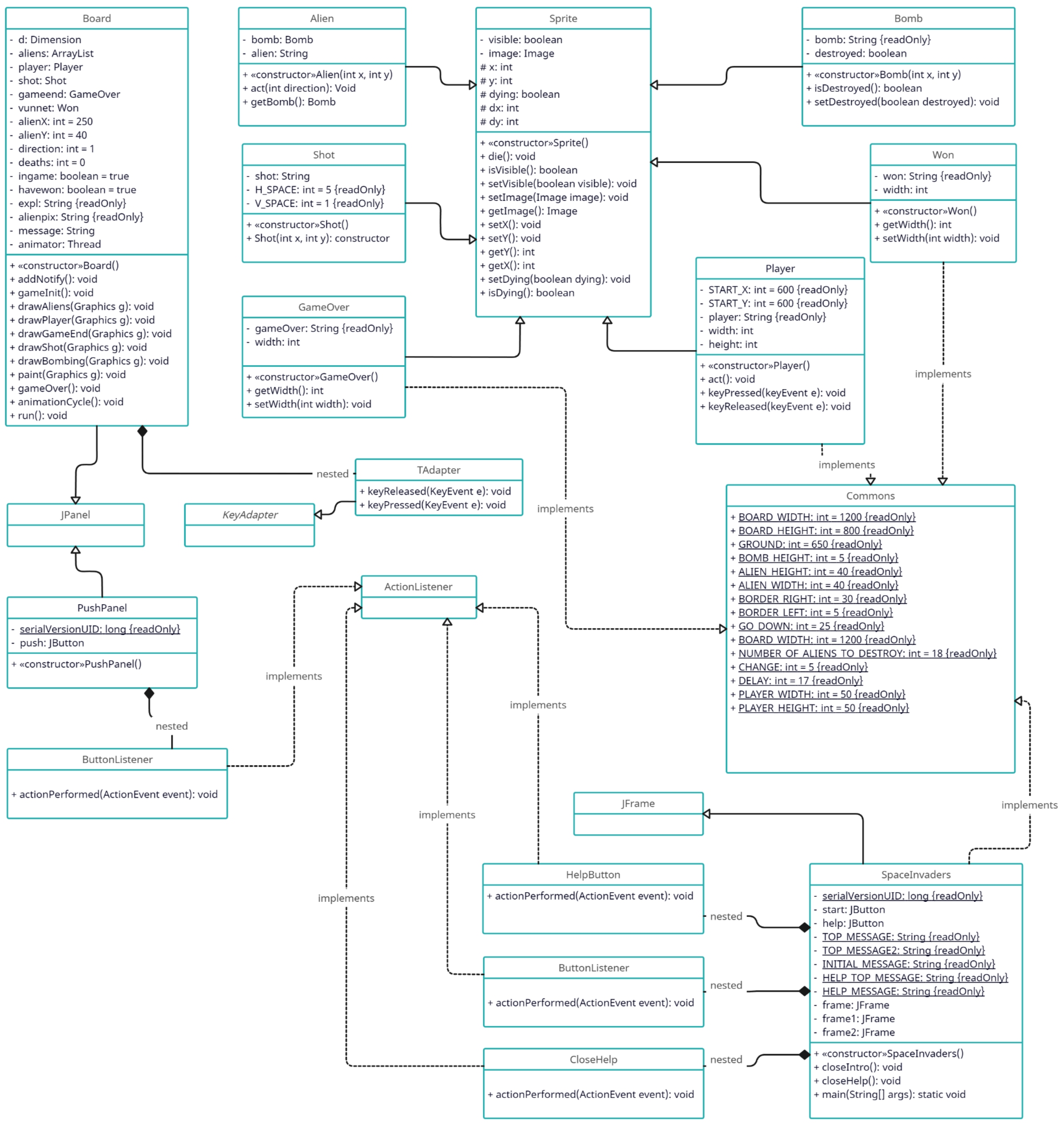
Won Class:

- **Won():**

- Purpose: Constructor to initialize properties of the victory screen.
- Logic:
  Set the image and initial coordinates.

3. UML Diagram

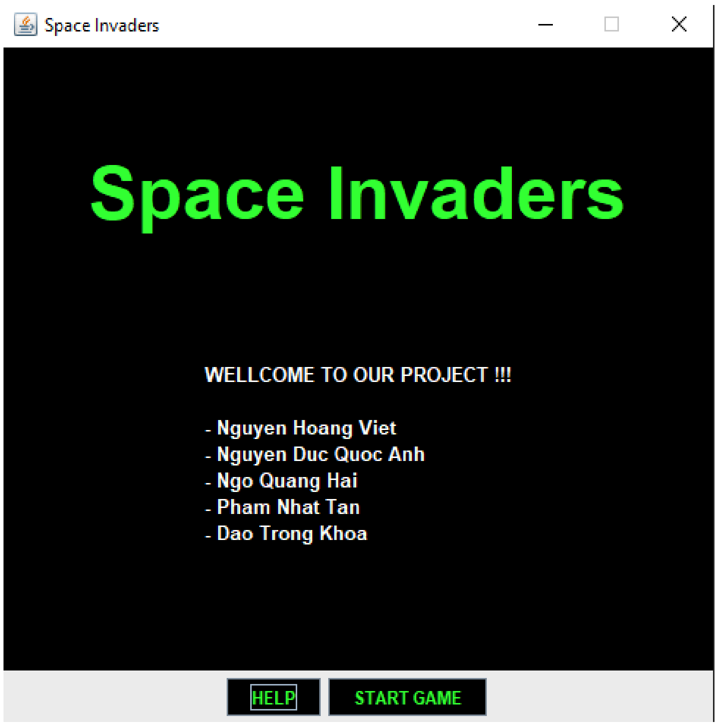
For a better visual representation of the project, bringing the more simple way of understanding the flow of the project, we composed a UML diagram for the whole project.


Board
- d: Dimension
- aliens: ArrayList
- player: Player
- shot: Shot
- gameend: GameOver
- vunnet: Won
- alienX: int = 250
- alienY: int = 40
- direction: int = 1
- deaths: int = 0
- ingame: boolean = true
- havewon: boolean = true
- expl: String {readOnly}
- alienpix: String {readOnly}
- message: String
- animator: Thread
+ «constructor»Board()
+ addNotify(): void
+ gameInit(): void
+ drawAliens(Graphics g): void
+ drawPlayer(Graphics g): void
+ drawGameEnd(Graphics g): void
+ drawShot(Graphics g): void
+ drawBombing(Graphics g): void
+ paint(Graphics g): void
+ gameOver(): void
+ animationCycle(): void
+ run(): void
Alien
- bomb: Bomb
- alien: String
+ «constructor»Alien(int x, int y)
+ act(int direction): Void
+ getBomb(): Bomb
Sprite
- visible: boolean
- image: Image
# x: int
# y: int
# dying: boolean
# dx: int
# dy: int
+ «constructor»Sprite()
+ die(): void
+ isVisible(): boolean
+ setVisible(boolean visible): void
+ setImage(Image image): void
+ getImage(): Image
+ setX(): void
+ setY(): void
+ getY(): int
+ getX(): int
+ setDying(boolean dying): void
+ isDying(): boolean
Bomb
- bomb: String {readOnly}
- destroyed: boolean
+ «constructor»Bomb(int x, int y)
+ isDestroyed(): boolean
+ setDestroyed(boolean destroyed): void
Shot
- shot: String
- H_SPACE: int = 5 {readOnly}
- V_SPACE: int = 1 {readOnly}
+ «constructor»Shot()
+ Shot(int x, int y): constructor
Won
- won: String {readOnly}
- width: int
+ «constructor»Won()
+ getWidth(): int
+ setWidth(int width): void
GameOver
- gameOver: String {readOnly}
- width: int
+ «constructor»GameOver()
+ getWidth(): int
+ setWidth(int width): void
Player
- START_X: int = 600 {readOnly}
- START_Y: int = 600 {readOnly}
- player: String {readOnly}
- width: int
- height: int
+ «constructor»Player()
+ act(): void
+ keyPressed(keyEvent e): void
+ keyReleased(keyEvent e): void
implements
implements
nested
TAdapter
+ keyReleased(KeyEvent e): void
+ keyPressed(KeyEvent e): void
implements
JPanel
KeyAdapter
Commons
+ BOARD_WIDTH: int = 1200 {readOnly}
+ BOARD_HEIGHT: int = 800 {readOnly}
+ GROUND: int = 650 {readOnly}
+ BOMB_HEIGHT: int = 5 {readOnly}
+ ALIEN_HEIGHT: int = 40 {readOnly}
+ ALIEN_WIDTH: int = 40 {readOnly}
+ BORDER_RIGHT: int = 30 {readOnly}
+ BORDER_LEFT: int = 5 {readOnly}
+ GO_DOWN: int = 25 {readOnly}
+ BOARD_WIDTH: int = 1200 {readOnly}
+ NUMBER_OF_ALIENS_TO_DESTROY: int = 18 {readOnly}
+ CHANGE: int = 5 {readOnly}
+ DELAY: int = 17 {readOnly}
+ PLAYER_WIDTH: int = 50 {readOnly}
+ PLAYER_HEIGHT: int = 50 {readOnly}
ActionListener
PushPanel
- serialVersionUID: long {readOnly}
- push: JButton
+ «constructor»PushPanel()
implements
implements
nested
ButtonListener
+ actionPerformed(ActionEvent event): void
implements
JFrame
implements
implements
HelpButton
+ actionPerformed(ActionEvent event): void
nested
SpaceInvaders
- serialVersionUID: long {readOnly}
- start: JButton
- help: JButton
- TOP_MESSAGE: String {readOnly}
- TOP_MESSAGE2: String {readOnly}
- INITIAL_MESSAGE: String {readOnly}
- HELP_TOP_MESSAGE: String {readOnly}
- HELP_MESSAGE: String {readOnly}
- frame: JFrame
- frame1: JFrame
- frame2: JFrame
+ «constructor»SpaceInvaders()
+ closeIntro(): void
+ closeHelp(): void
+ main(String[] args): static void
ButtonListener
+ actionPerformed(ActionEvent event): void
nested
CloseHelp
+ actionPerformed(ActionEvent event): void
nested

UML Diagram for the whole project

CHAPTER 3: DEMO - RESULT

In order to run our attempt-on-recreating the classic Space Invaders, a device with IDE and Java Development kit installed is required. Below are the samples taken directly from the run programme, representing the current state of the project.

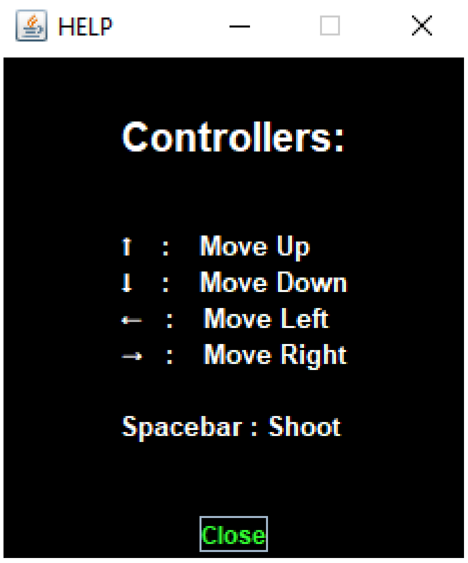


Space Invaders

WELLCOME TO OUR PROJECT !!!

- Nguyen Hoang Viet
- Nguyen Duc Quoc Anh
- Ngo Quang Hai
- Pham Nhat Tan
- Dao Trong Khoa

The entry menu screen from the game

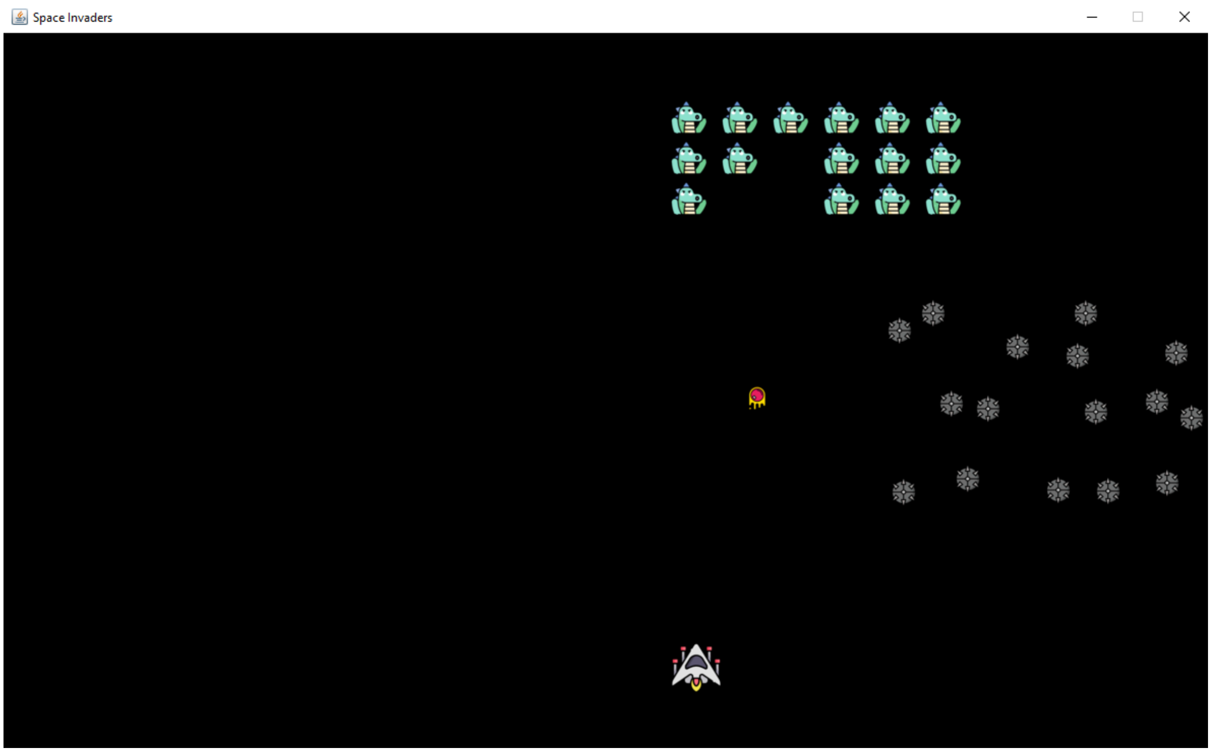


Controllers:

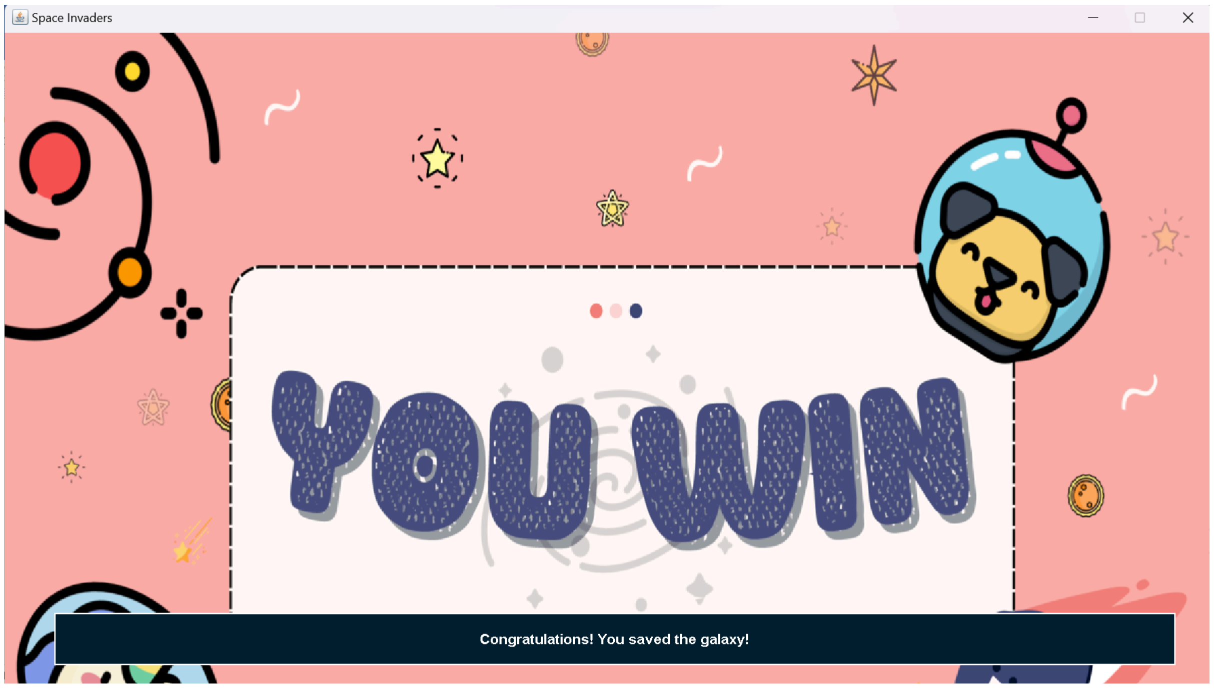
↑ : Move Up
↓ : Move Down
← : Move Left
→ : Move Right

Spacebar : Shoot

Help menu

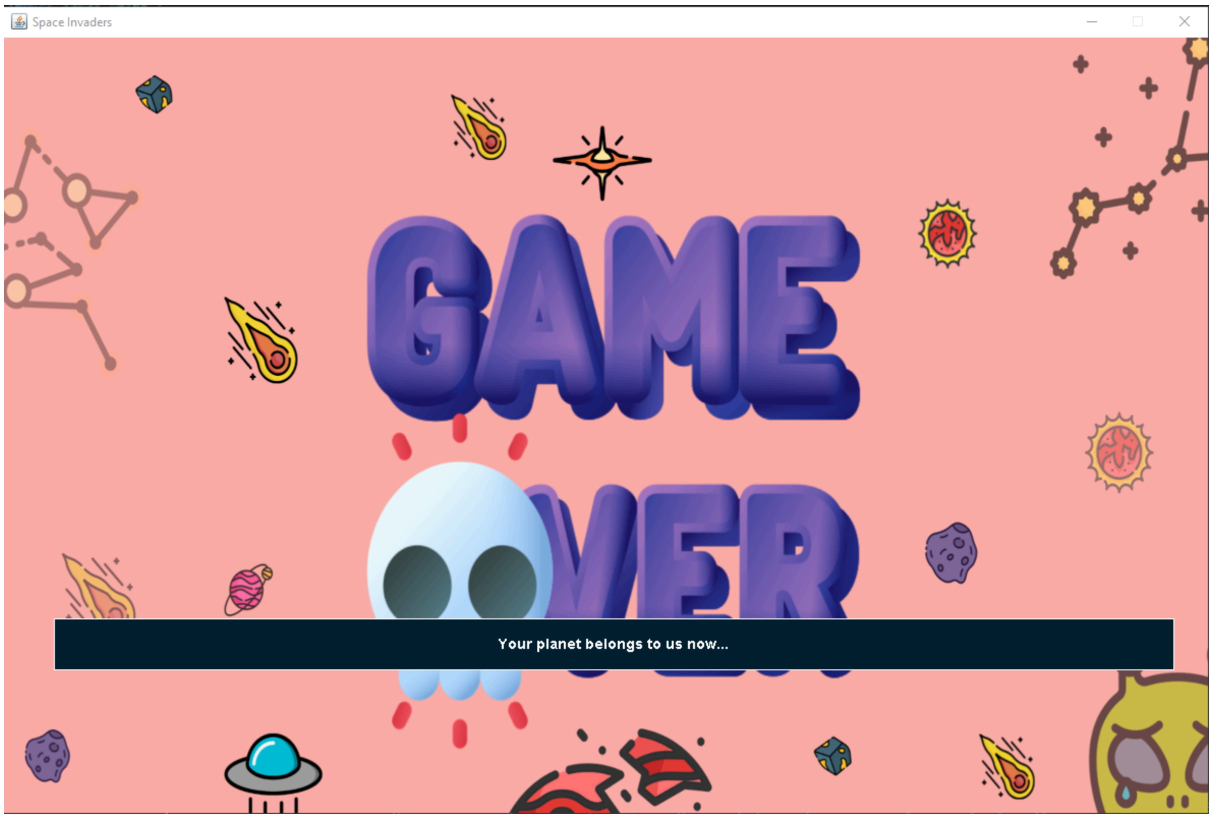


Gameplay



Congratulations! You saved the galaxy!

Victory Screen



Your planet belongs to us now...

Game Over screen

CHAPTER 4: CONCLUSION & FUTURE WORKS

1.conclusion

After a course of Object Oriented Programming, our team has not only studied and dived deeper to the world of OOP, but by working on this project parallelly also grant us more practical and understanding point of view toward the subject, bringing us to this goal of recreating the functional programme of a classic piece of media in the pop culture.
Apart from improved programming development skills, we also learn to interact and use github, a tool which would play an important role in the coding field, effectively.
We also learnt about workload division, and shared the work into different parts that fit each and every member.

After this project, our team would eventually move on to pursue bigger and much more complex projects, but thanks to the Space Invaders, our very first joint project, we already can see the vision as well as being familiar with the coding language and the workflow as a team of game developers.

2. Future Works

Though our Space Invaders features almost every essential part of the original game, our team all agreed that there is more than enough room for development within this project. Initially Space Invaders is a fixed shooter game where objects like players can only move horizontally , shot and waves of aliens move vertically. So we aimed to develop this feature so that every object from the game can move freely, giving it a more unexpected factor toward players.

We also learnt from many modern implementations of Space Invaders, such as Chicken Invaders. Where not only the object can move to any direction in given patterns, players can get to experience different options of playing style: different types of bullets, different types of enemies, boss fights,... in which we were so ambitious about. Unfortunately, we cannot input any of those bigger ideas into the project in such a short time. So any further commitments from any members are therefore very appreciated.

3.Acknowledgement
We would like to give our sincerest thanks and show our utmost appreciation toward our lecturer and people that assist us through-out this period in order to achieve the project's goal:

- Dr. Tran Thanh Tung
- Mr. Phu
- And five members of our team to dedicatedly crafted this project together

REFERENCE:

https://github.com/yenthehaiquang/OOP-Project-SpaceInvaders-Game
https://zetcode.com/javagames/spaceinvaders/
https://github.com/apetenchea/SpaceInvaders/blob/master/README.md
https://github.com/howuvebeen/space-invader
https://www.perlmonks.org/?node_id=171425
https://www.linkedin.com/pulse/software-development-space-invaders-remake-java-oop-nick