INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

PROJECT REPORT

GRYBOTECTOR

OBJECT-ORIENTED PROGRAMMING (IT069IU)

Course by

Dr. Tran Thanh Tung and MSc. Pham Quoc Son Lam

VORTEX SQUAD MEMBERS:	**Students' ID**
Nguyễn Đức Tâm	ITCSIU21031
Ngô Vũ Cao Long	ITCSIU21085
Lương Anh Vũ	ITITIU21351
Nguyễn Đặng Thành Duy	ITITIU21191
Nguyễn Anh Thắng	ITCSIU21233

Table of Contents

Figure List

CONTRIBUTION TABLE

No.	Full name	Students' ID	Contributor
1	Nguyễn Đức Tâm	ITCSIU21031	20%
2	Ngô Vũ Cao Long	ITCSIU21085	20%
3	Lương Anh Vũ	ITITIU21351	20%
4	Nguyễn Đặng Thành Duy	ITITIU21191	20%
5	Nguyễn Anh Thắng	ITCSIU21233	20%

ABSTRACT

---*****---

Grybotector is a 2D shooting game developed as a project of Object-Oriented Programming at HCMIU VNU. The game is set in a futuristic world where monsters have taken over the planet and the player is tasked with defeating them to save humanity.

In the game, the player takes on the role of a Grybotector, a special agent equipped with advanced weaponry and a set of skills to fight against the monster invasion. The game features various levels, each with unique challenges and objectives that the player must complete to progress to the next level.

The gameplay of Grybotector is fast-paced and action-packed, with the player battling against waves of enemy robots using a variety of weapons such as pistols, shotguns, and rifles. The game also includes power-ups and upgrades that the player can collect to enhance their abilities and weapons.

The graphics of Grybotector are vibrant and colorful, with detailed environments and character designs. The sound effects and music add to the immersive experience, providing an adrenaline-pumping soundtrack that complements the action on screen.

Overall, Grybotector is an exciting and challenging shooting game that showcases the skills and creativity of the developers at HCMIU VNU. With its engaging gameplay and impressive graphics, the game is sure to provide hours of entertainment for players of all ages.

Keywords: Grybotector, shooting, game, adventure,Object-Oriented Pragramming.

INTRODUCTION

---*****---

1. Objectives:

Grybotector is a 2D pixel-based shooting game, developed with the goal of mastering Java programming and implementing OOP and SOLID design principles while creating an enjoyable and accessible gaming experience. The game's premise features a soldier in a jungle environment, battling enemies in intense shooting encounters. The project aims to enhance proficiency in Java, particularly Java Swing, and apply OOP and SOLID principles to foster modular and maintainable game architecture. Additionally, the game intends to appeal to a diverse audience, providing entertainment while honing development skills. Grybotector showcases the team's commitment to technical excellence and artistic entertainment for players of all backgrounds.

Summary of our objectives:

- Creating a shooting game to learn and practice using Java language.
- Applying OOP paradigm in coding progress.
- Observing SOLID rules to have clean code.
- Engaging in the iterative process of game administration and code optimization.
- Assessing the capacity to incorporate additional functionalities onto the fundamental codebase.
- Acquiring the skills of project configuration, programming, and collaborative teamwork within a multifaceted programming endeavor comprising multiple participants and diverse programming components.

Figure 1: Grybotector banner.

2. The tools used:

- IDE for programming and debugging: JetBrains IntelliJ, Visual Studio Code.
- Mean of code version management: GitHub.
- Means of contacting: Trello and OneDrive.
- Character design and image editing: Adobe Photoshop, Microsoft Paint.
- Map design and creation: Map editor.
- Image composition and sprite creation: Sprite sheet packer.
- Sprite sheet Splitting or Sprite Sheet Decomposition: Image splitter.
- Image Resizing or Image Scaling: Crop image, Resize picture.
- Animated Sprite editor and Pixel Art tool: Aseprite.

METHODOLOGY

---*****---

1. Rules:

- The main character you will embody is *a soldier*.
- Your mission is to eliminate all the monsters you encounter while defeating the boss invading the Earth.
- The main character will *be controlled by the player*.
- The main character will have a total of *5 health points*.
- *Each bullet shot* by the main character will *cause the monster to lose 1 health point*.
- When *falling into the water*, the main character will *lose all health points* and *start the game again* from the beginning.
- *Move forward and shoot, defeating monsters* is all you need to do.
- There are *4 types of monsters* fighting against the player:
    - *Ghost monsters*
    - *Stone monsters*
    - *Demon king boss*
- Each type of monster has different fighting styles and health points:
    - *Ghost monsters:* stand still, deal 2 damage, and have 5 health points.
    - *Stone monsters:* have a relatively short attack range, deal 1 damage, and have 8 health points.
    - *Demon king boss:* has a short attack range, deals 3 damage, and has 10 health points.
- When *approaching any monster*, the main character will *lose 1 health point*.
- The game consists of *2 stages*:
    - *State 1* includes *small monsters*, overcoming all obstacles, and reaching the gate leading to *stage 2*.
    - *Stage 2* includes small monsters and *Demon king boss*
    - When you *defeat the big boss*, you will *win the game*.
- When the main character *runs out of health points*, he will *die* and the game will *restart from the beginning*.

2. Game instruction:

- Press the left and right buttons to move the character.
- Press the down button to make the character kneel and move slower, as well as dodge bullets from enemies.
- Press the K key to shoot.

- Each bullet that hits an enemy will cause them to lose 1 health.
- When their health reaches zero, they will be defeated.
- You will lose 1 health if you get close to an enemy or get hit by their bullets.
- When your health reaches zero, your main character will die, and you will restart from the beginning.
- Defeat the boss to proceed to the next level.

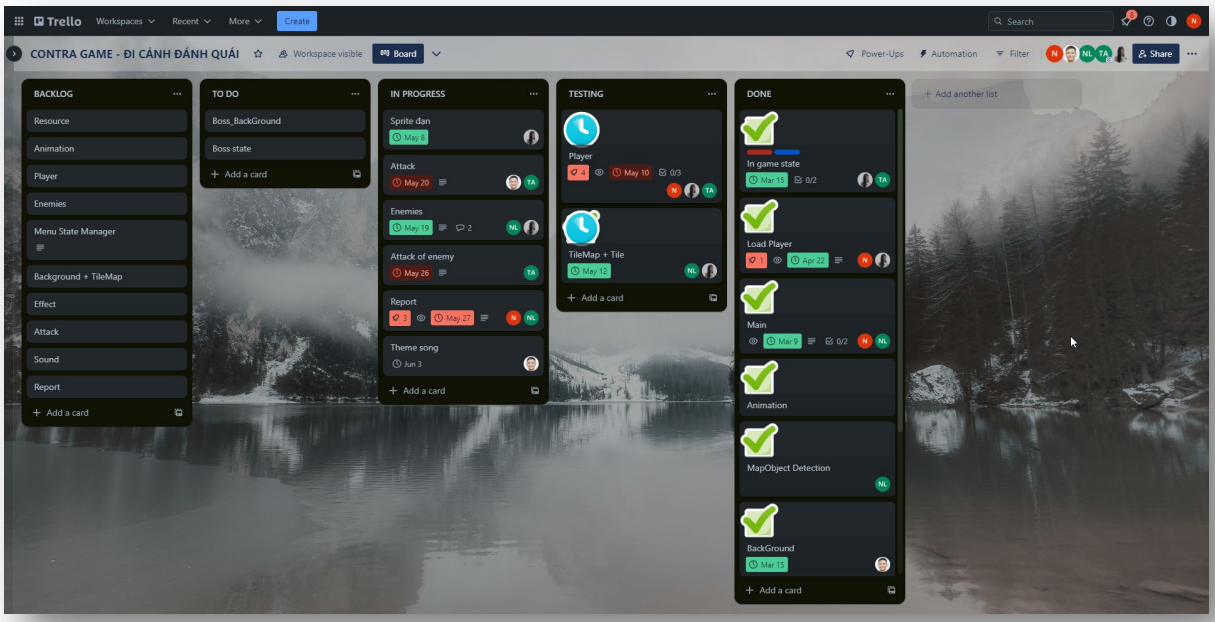

Figure 2: Trello work space

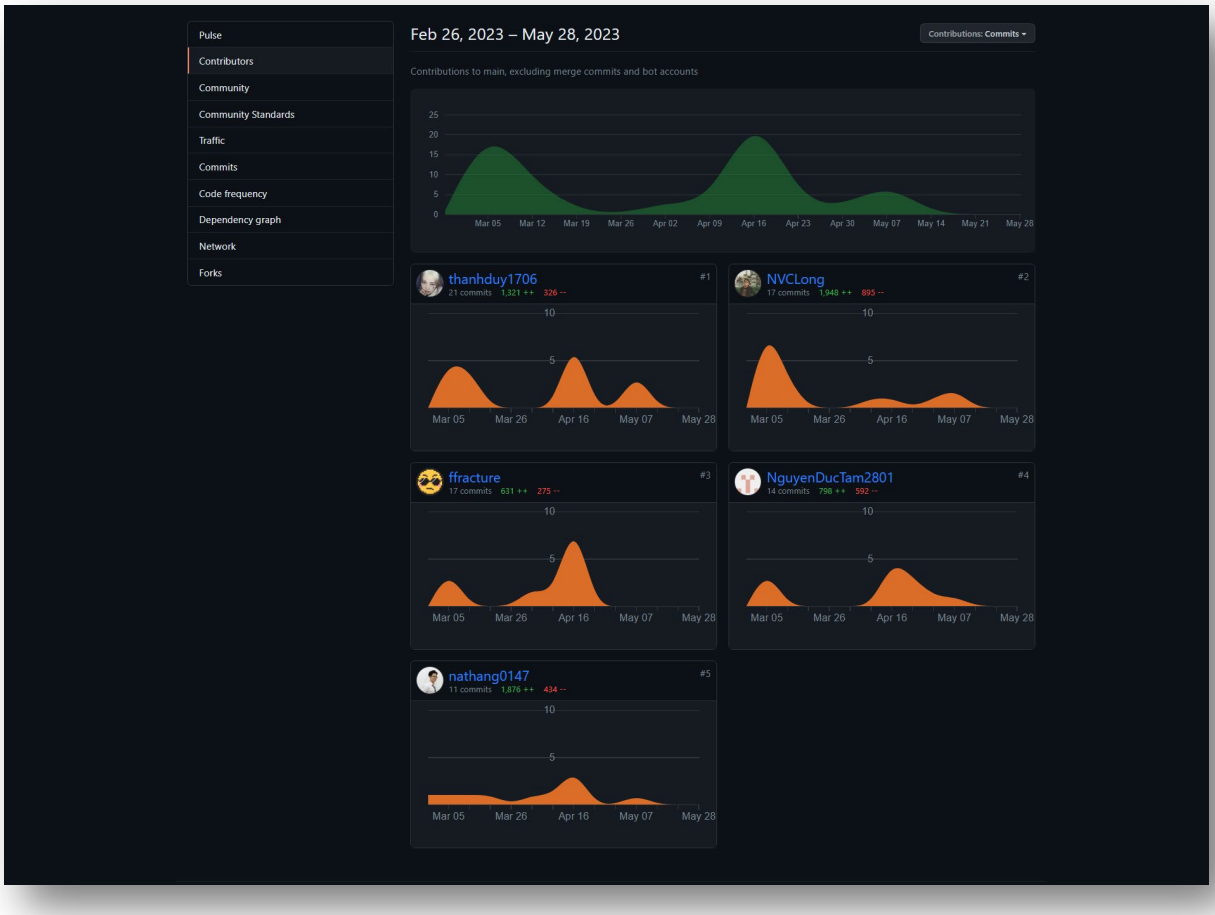

Figure 3: GitHub statistic

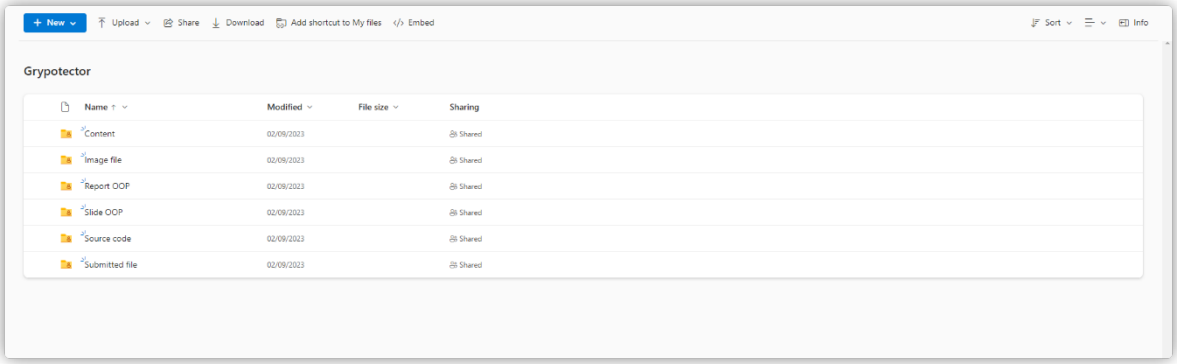

Figure 4: OneDrive file structure

3. Designs:

a) *UI/UX:*

Prior to engaging in gameplay, we have consistently prioritized UI/UX, thus drawing significant inspiration from external sources and crafting our own unique layout.

Taking inspiration from the adventure theme, we directed our attention towards transforming key aspects. This included replacing the typical street setting with a dense jungle backdrop, introducing an engaging shooting game type, and featuring a courageous soldier as the protagonist who is fighting against various dangerous and evil enemies. To create the suitable theme design, we must experience 3 steps, and they are:

1) Researching and meticulously selecting appropriate environments for the game, encompassing backgrounds, main character visuals, creatures, and various other integral elements.

2) Establishing standardized templates for the components of the game and redesigning visuals to align with the established guidelines.

3) Through iterative feedback from team members and continuous modifications, we relentlessly pursued the optimal combinations of colors and design patterns.

In specific, each component we have redesigned includes the following:

- *Background:*

Figure 5: Background Stage 1

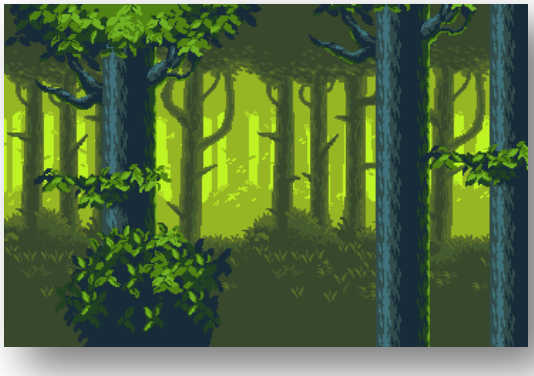

Figure 6: Background Stage 2

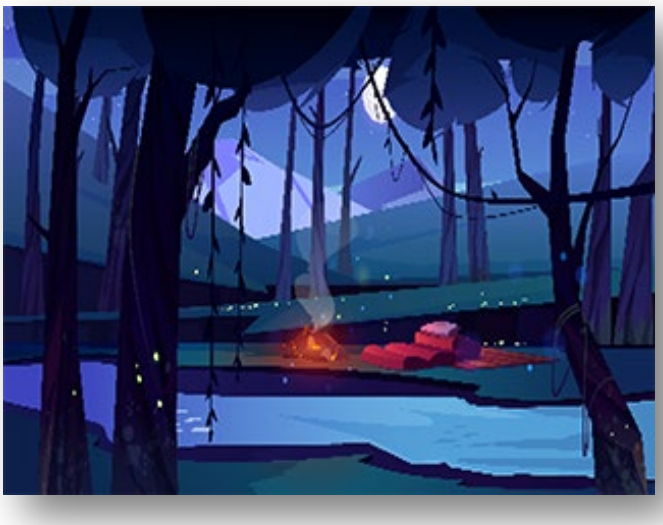

Figure 7: Background Menu

- *Main character:*

Figure 8: Sprite of main character

- *Enemies:*

Figure 9: Sprite of enemy 1

Figure 10: Sprite of enemy 2

Figure 11: Sprite of Boss

- *Maps:*

Figure 12: Map 1

Figure 13: Map 2

- *Bullets:*

Figure 14: Sprite of bullet

- *Menu buttons:*

Figure 15: Menu buttons

- *Accomplished board:*

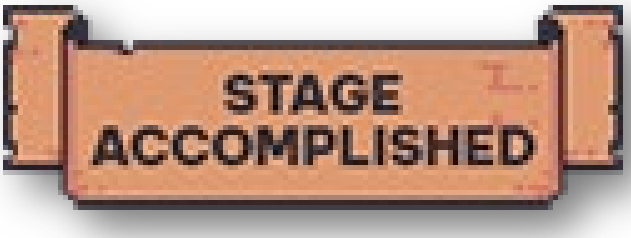

Figure 16: Stage accomplished board

- Defeat Incurred image:

Figure 17: Defeat Incurred

- *Icon image:*

Figure 18: Icon image

- *Game paused board:*

Figure 19: Game paused board

- *Game paused buttons:*

Figure 20: Game paused buttons

- *Health:*

Figure 21: Health bar

- *Tile sheet:*

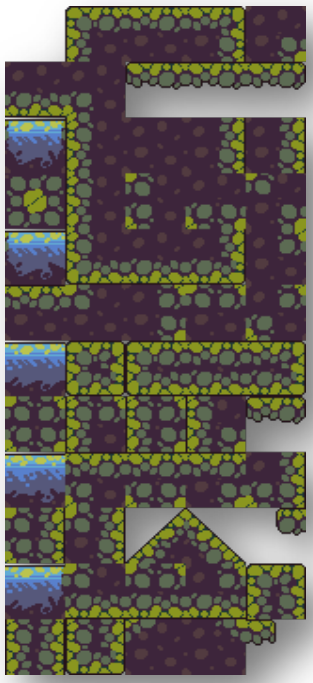

Figure 22: Tile sheet

Putting them together with some parts taken online:

- Sound
- Buttons

In order to ensure the intended gameplay experience, it is imperative that the client's screen resolution adheres to the specified dimensions of 320x240. Any deviation from this requirement may result in game instability or malfunctions. Moreover, given our limited expertise in the realm of user experience (UX), we dedicated significant attention to this aspect. Our efforts focused on meticulously crafting intuitive user actions, which were successfully implemented:

- The character's motion and animation system.
- The diverse character system.
- The complexity of the map structure.
- The difficulty level of the game stages and the gameplay mechanics.
- The graphics of the game.
- The depth of the storyline manifested through the game stages.

Despite our best efforts, certain game functions may not operate as seamlessly as originally anticipated. However, we remain committed to conducting further research in order to enhance the UI/UX of this game in future iterations.

b) Game Loop:

We have referenced the loops and games of the same genre such as shooting, adventure, etc. on the internet and have been able to create a loop for our game with the following diagram:

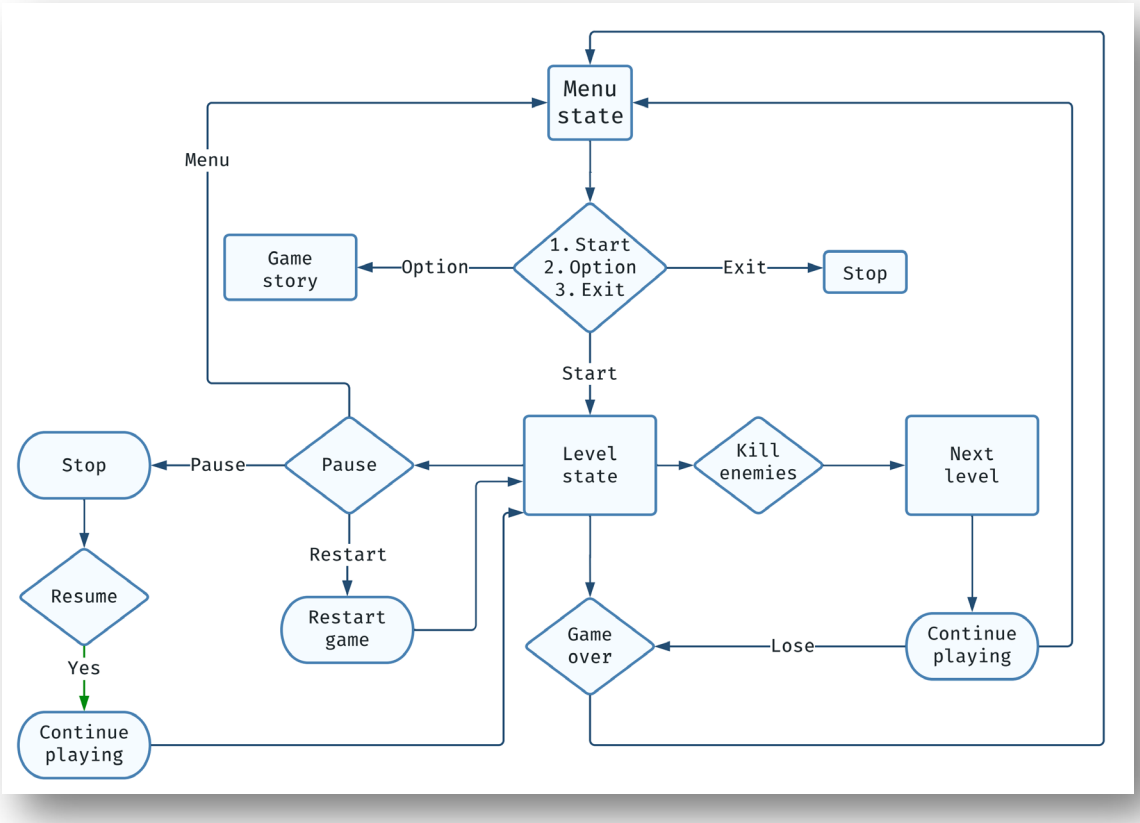

Figure 23: Game Loop diagram

c) *Design pattern:*

We use the Singleton design pattern allows creating a class with only one instance and provides global access to that instance. The applications of the Singleton pattern include:

- Creating a global configuration class to hold configuration settings for the entire application.
- Managing database connections to ensure only one connection exists and is accessed from different parts of the application.
- Implementing a caching mechanism to access and update cached storage data from different parts of the application.
- Logging capabilities to allow access and writing of log entries from different parts of the application to the same log file or system.
- Using in a multi-threaded environment to ensure safe and unique access to shared resources.

This is my code that apply Singleton design pattern:

```
31      private static GamePanel gp;
        1 usage  Lương Anh Vũ +2
32      private GamePanel() {
33          super();
34          setPreferredSize(new Dimension( width: WIDTH * SCALE,  height: HEIGHT * SCALE));
35          setFocusable(true);
36          requestFocus();
37      }
        1 usage  Long Ngo
38      public static GamePanel getPanel(){
39          if(gp==null){
40              gp= new GamePanel();
41          }
42          return  gp;
43      }
```

Figure 24: Singleton Design pattern

d) Game algorithm:

After investing a significant amount of time in code repetition and debugging, we have successfully achieved the final project structure, as depicted in the image below.

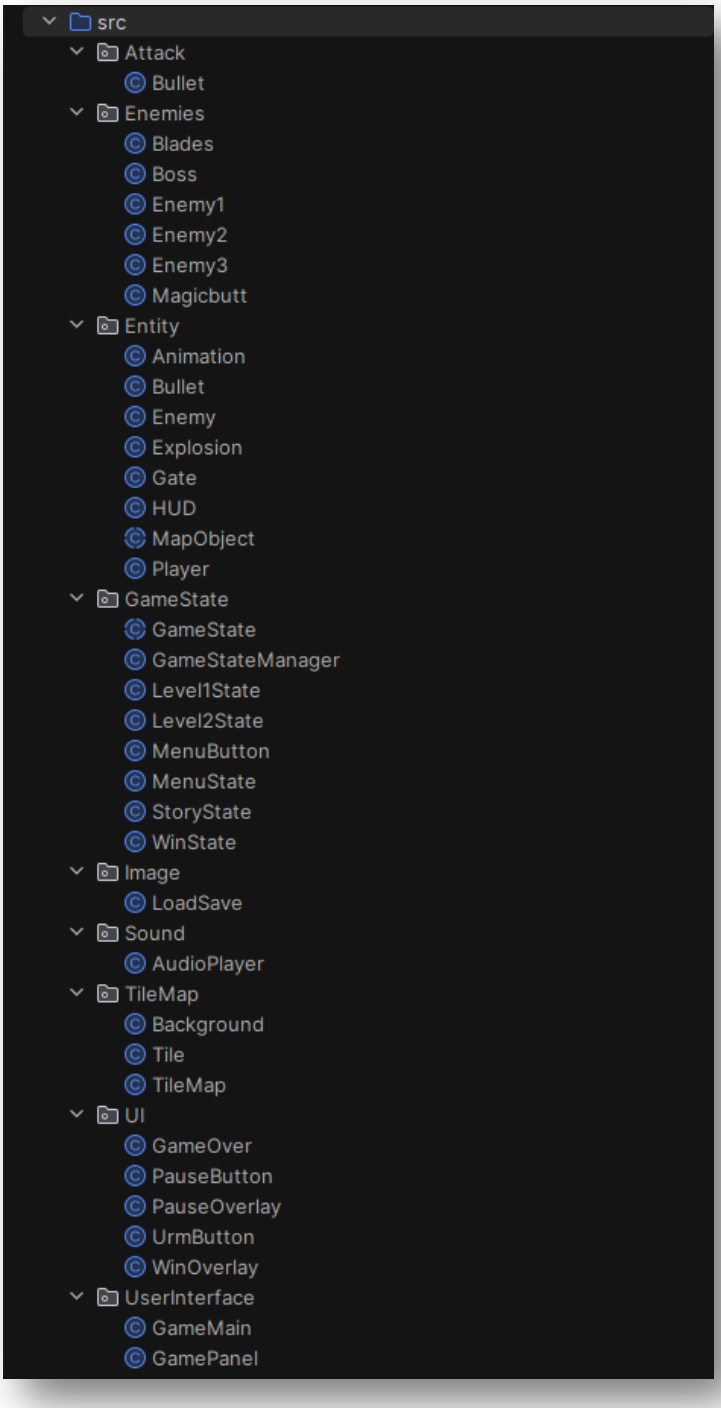

Figure 25: Project structure

We can group our classes into a specific group, such as:

- UserInterface: The primary approach in this game involves initializing a screen, which is made by *GamePanel* extending *JPanel*, in *GameMain* class.

- UI:

- The *PauseButton* class represents a pause button in the UI. It has attributes for x, y, width, and height that define its position and size. The createBounds() method creates a rectangle that represents the button's bounds. This class provides a foundation for creating interactive buttons with defined boundaries.

- The *PauseOverlay* class manages the pause menu overlay. It has a backgroundImg attribute representing the background image of the pause menu. The update() method updates the menu buttons based on the current choice. The draw() method is responsible for rendering the overlay and buttons on the screen. It uses instances of the *UrmButton* class to handle the menu, replay, and unpause buttons.

- The *UrmButton* class extends the PauseButton class and represents a specific type of button in the pause menu. It loads images for different button states using the loadImgs() method. The update() method adjusts the button's appearance based on the current choice, highlighting it if selected. The draw() method renders the button on the screen. Additional attributes like keyOver and keyPressed track user interactions with the button.

- TileMap:

- The *Background* class manages and displays the background image of the tile map, allowing for positioning and movement.
- The *Tile* class represents individual tiles in the tile map, storing their image and type (such as "NORMAL" or "BLOCK").

- The *TileMap* class handles the overall functionality of the tile map, including loading tile set and map data, managing position and boundaries, and rendering the tiles on the screen.

    - Sound: only consists of class *ThemeSong*. It takes a file location as a parameter and uses the AudioInputStream class to read the audio data from the specified file. If the file exists, it creates a Clip object and opens it for playback.

    - Image: The *LoadSave* class in the Image package is responsible for loading and saving game assets related to images. It includes a method called GetSpriteAtlas that takes a file name as input and returns a BufferedImage object representing the sprite atlas. The class also provides constant strings for the file paths of menu buttons (MENU_BUTTONS) and the menu background (MENU_BACKGROUND). The GetSpriteAtlas method loads the image using ImageIO.read and returns it.

    - GameState:

- *GameState* class: serves as the base class for all game states. It contains an attribute called gameStateManager, which is a reference to the *GameStateManager* class. This allows the *GameState* objects to communicate with the game state manager and perform state transitions. The *GameState* class also includes methods such as init(), update(), and render() to initialize, update, and render the game state, respectively. These methods provide a common interface for all game states to implement their specific functionality. Additionally, the *GameState* class may include attributes like inputManager or entityManager to manage user input or game entities, depending on the requirements of each state.

- *GameStateManager* class: is responsible for managing and coordinating different game states. It maintains an ArrayList of *GameState* objects to store and manage the various states. The *GameStateManager* class includes methods like pushState(), popState(), and setState() to add new states, remove

states, and set the current state of the game. These methods facilitate state transitions and allow the game to progress from one state to another. The class also has an attribute called currentState, which keeps track of the current state of the game. By updating and rendering the current state in its own update() and render() methods, the GameStateManager ensures that the appropriate state logic and visuals are applied.

- *Level1State* class: represents a specific level of the game. It extends the *GameState* class and includes methods like init(), update(), and render(), which provide the level-specific initialization, updating, and rendering functionality. For example, the *Level1State* class may contain attributes like enemyManager, powerUpManager, and obstacleManager to manage the enemies, power-ups, and obstacles specific to this level. The class may also include methods like handleCollisions() or checkWinCondition() to handle the level-specific collision detection or victory conditions.

- *Level2State* class: represents another level of the game. It extends the *GameState* class and includes methods like init(), update(), and render() for level-specific initialization, updating, and rendering. This class may have its own set of attributes and methods to manage the entities and logic specific to this level. For instance, it may include bonusEnemyManager, specialPowerUpManager, and complexObstacleManager to handle the unique entities and obstacles found in the second level. Level-specific collision detection or victory conditions can also be implemented in this class.

- *MenuButton* class: represents a menu button entity used in the menu state. It may have attributes such as position, size, and text to define the button's visual appearance and properties. The class includes methods like onClick() or onHover() to handle user interactions with the button. For instance, the onClick() method can be implemented to perform a specific action when the button is clicked, such as transitioning to a new game state. The *MenuButton* class may also include methods like render() to display the button on the menu screen.

- *MenuState* class: represents the menu screen in the game. It extends the *GameState* class and includes methods like init(), update(), and render() for menu-specific initialization, updating, and rendering. This class may contain attributes such as an ArrayList of *MenuButton* objects to store and manage the menu buttons available in the menu state. The *MenuState* class provides the functionality to navigate the menu options, select buttons, and trigger corresponding actions. It may include methods like handleInput() to detect user input and update the selected button based on the user's interaction. The *MenuState* class works in conjunction with the *MenuButton* class to create an interactive menu system for the game.
    - Entity:
    - Enermies:
    - Attack:

4. UML Diagram:

To enhance the clarity of the project's structure and algorithms, we have incorporated UML diagrams for both the entire project and each mentioned group.

There are all UML diagrams for our project:

GRYBOTECTOR

UML DIAGRAM

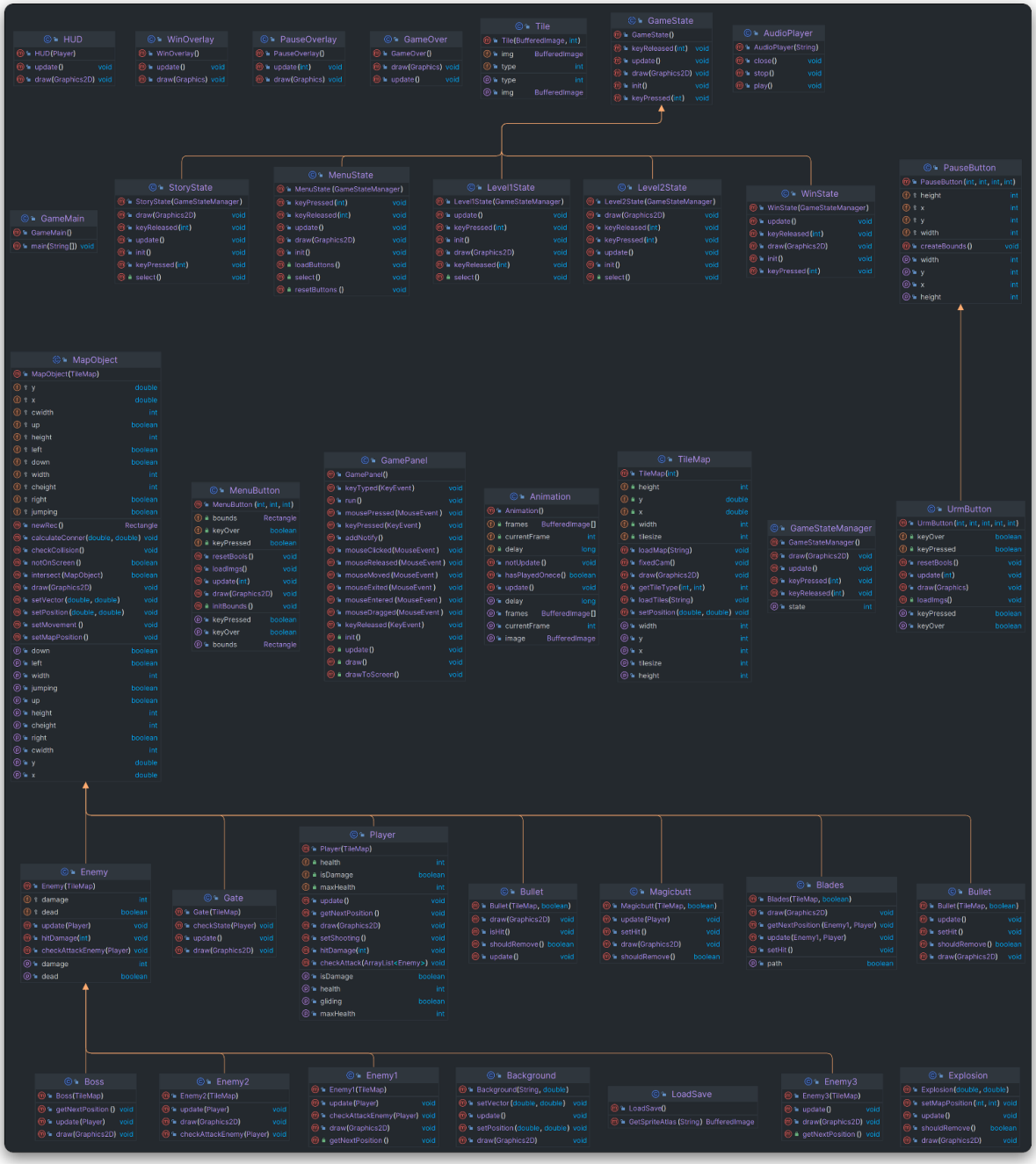

Figure 26: UML with whole project

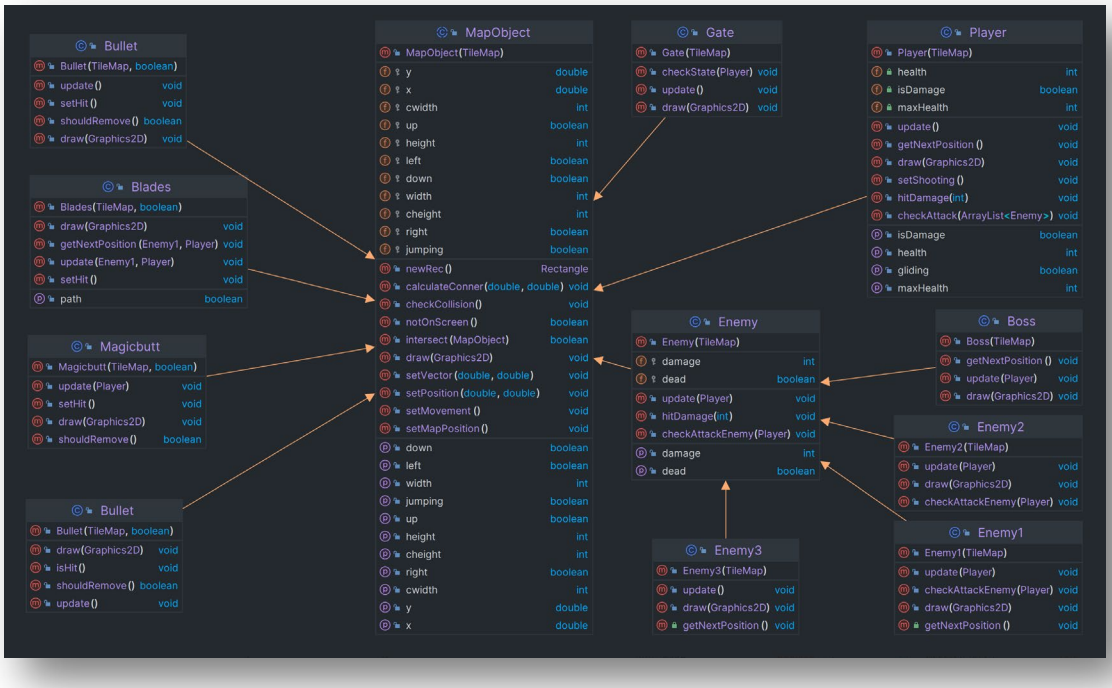

Figure 27: UML of Map Object

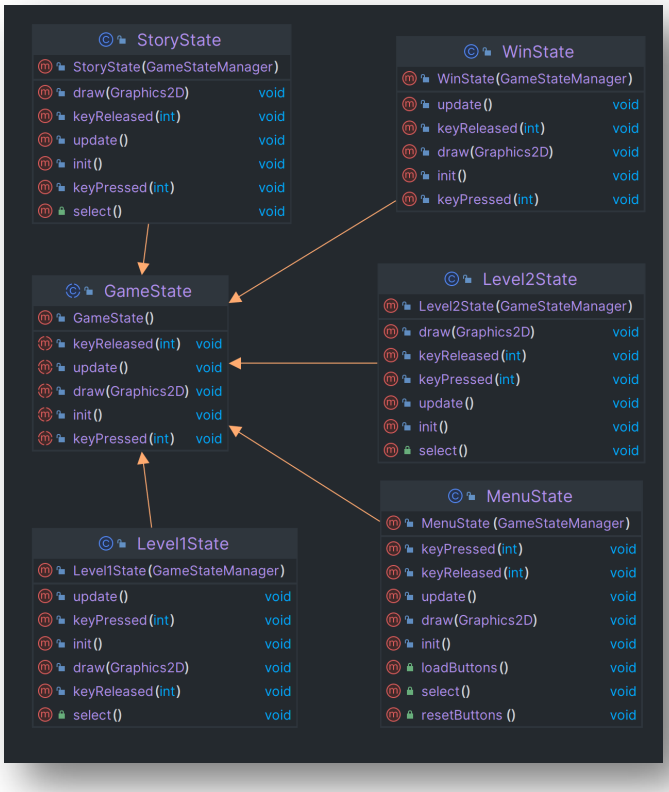

Figure 28: UML of Game State

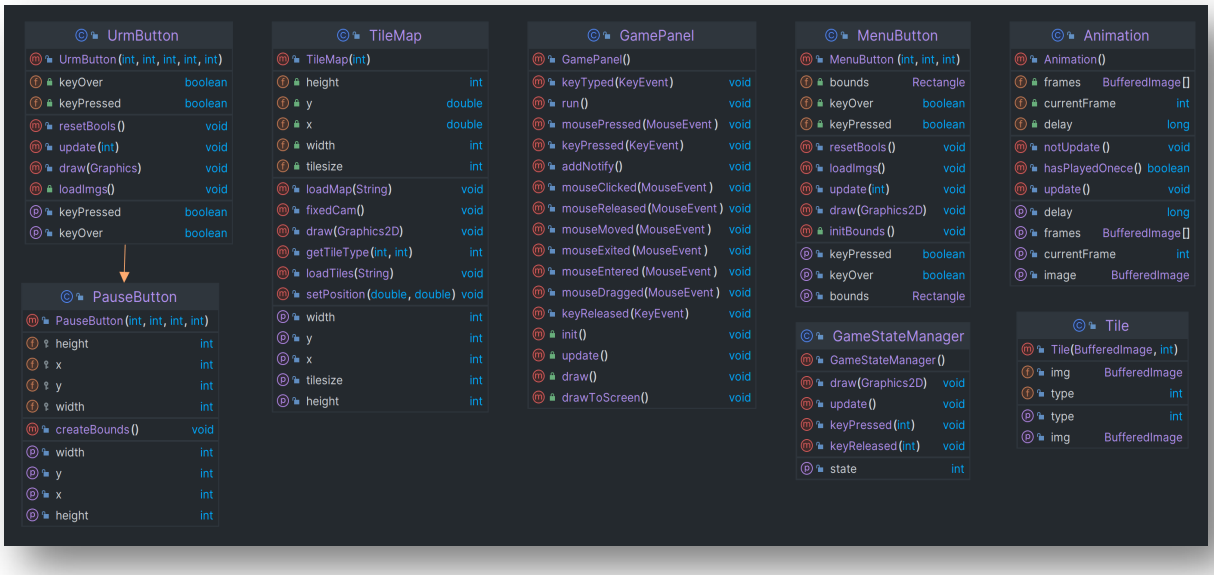

Figure 29: UML of Game UI

Figure 30: UML of Game UX

CHAPTER 3: RESULT

---*****---

In order to conduct testing of our game, we utilized a machine equipped with an IDE and Java Development Kit. We pulled the code from Git and compiled the Main class to initiate the game. Presented below is a sample of the game states that showcase the current build of the game.

Figure 31: Menu Screen

Figure 32: Game playing (State 2)

Figure 33: Demon King Boss

Figure 34: Game Paused Screen

Figure 35: Stage Accomplished

Figure 36: The Gateway to Stage 2

CHAPTER 4: CONCLUSION AND FUTURE UPDATE

---*****---

1. Conclusion:

The game's development remains ongoing, and in the final phase, the team has acquired a more comprehensive understanding of the four fundamental aspects of object-oriented programming (OOP) and the SOLID principle. This enhanced knowledge has greatly facilitated the seamless integration of OOP principles into game development and the programming workflow, resulting in the implementation of novel features not present in the original version. Encapsulation, a core concept, has been extensively explored across various project classes. Among them, the Card class has exemplified the frequent application of inheritance, abstraction, and polymorphism. As a testament to the team's commitment to adhering to the fundamental principles of OOP, the Grybotector game has been developed with utmost adherence to these principles, incorporating all four key OOP features and leveraging a design pattern acquired through the course. Recognizing the significance of this opportunity, the experienced team members are fully devoted to achieving its success, acknowledging that this endeavor may not present itself again.

2. Future works:

In the future, the game will undergo continuous updates as part of the ongoing development process. One of the key areas of focus will be the implementation of a buying and selling system for the main character, enabling players to engage in transactions and trade various items. Additionally, the game plans to introduce a dynamic element by incorporating the random dropping of items from alternative objects such as wood boxes, rocks, and more, adding an element of surprise and variety. Furthermore, efforts will be made to enhance the characters' system, including their abilities, customization options, and overall progression within the game. Another exciting addition to the game will be the creation of various power gun types for our soldier, providing players with a diverse arsenal to choose from and allowing for strategic gameplay. These future developments aim to enrich the gaming experience, ensuring continuous engagement and enjoyment for the players.

3. Acknowledgment:

We extend our deepest gratitude to our lecturer and all the individuals who have provided invaluable assistance in helping us accomplish the objectives of this project:

- Dr. Tran Thanh Tung
- MSc. Pham Quoc Son Lam
- Original code from pawelpaszki (Paszki, 2015/2021)
- The sites Geeksforgeeks, Javapoints, and so on
- The README.md template from othneildrew (Drew, 2018/2022)

REFERENCES

---*****---

- Night Forest with Camp Fire, River, Mountains:

This is a free vector illustration depicting a night forest scene with a campfire, river, and mountains. It can be used as a background or part of the game environment.

Link: Night Forest with Camp Fire, River, Mountains

- Free Swamp 2D Tileset (Pixel Art):

This is a free tileset consisting of pixel art graphics designed for creating swamp-themed game environments. It includes various elements such as terrain tiles, trees, rocks, and more.

Link: Free Swamp 2D Tileset (Pixel Art)

- Complete GUI Essential Pack:

This pack offers a comprehensive set of graphical user interface (GUI) elements for game development. It includes buttons, checkboxes, sliders, progress bars, and other UI components in a modern and cohesive design.

Link: Complete GUI Essential Pack

- Fire Pixel Bullet 16x16:

This asset provides a pixel art representation of a fire-themed bullet projectile. It is designed in a 16x16 pixel size and can be used for various shooting or combat effects in a game.

Link: Fire Pixel Bullet 16x16

- Free Swamp 2D Tileset (Pixel Art):

Same as the second link you provided, this is a free tileset with pixel art graphics specifically designed for creating swamp-themed game environments.

Link: Free Swamp 2D Tileset (Pixel Art)

- Team Wars Platformer Battle:

This is a platformer battle game asset pack that includes various character sprites, animations, backgrounds, and UI elements. It is suitable for creating a team-based platformer game.

Link: Team Wars Platformer Battle

- Mecha Golem Free:

This asset provides a pixel art representation of a Mecha Golem character. It includes different animations and can be used as an enemy or player character in a game.

Link: Mecha Golem Free

- Sunnyland Tall Forest:

This asset pack offers a set of pixel art graphics featuring a tall forest environment. It includes various elements such as trees, foliage, rocks, and more for creating immersive game levels.

Link: Sunnyland Tall Forest

- Sprout Lands UI Pack:

This UI pack provides a collection of graphical user interface (GUI) elements with a sprout lands theme. It includes buttons, checkboxes, progress bars, and other UI components designed in a cute and vibrant style.

Link: Sprout Lands UI Pack

- Basic Pixel Health Bar and Scroll Bar:

This asset provides basic pixel art representations of a health bar and a scroll bar. It can be used to display health or other progress indicators in a game.

Link: Basic Pixel Health Bar and Scroll Bar

- Rogue Knight:

Description: This asset is for a game called Rogue Knight. It provides pixel art graphics, including character sprites and animations, for a knight character in a platformer game.

Link: Rogue Knight

- Best README Template:

Description: This is a GitHub repository that provides a template for creating well-structured and informative README files for software projects. It offers guidelines and sections to include to ensure a comprehensive README.

Link: Best README Template

- Java Swing - Javatpoint:

Description: This is a tutorial page from Javatpoint that covers Java Swing, a GUI toolkit for Java applications. It provides explanations, examples, and code snippets to learn and understand Java Swing concepts.

Link: Java Swing – Javatpoint

- Singleton Class in Java - GeeksforGeeks:

Description: This is an article from GeeksforGeeks that explains the Singleton class design pattern in Java. It provides implementation details and examples to understand the concept of creating a class with only one instance.

Link: Singleton Class in Java – GeeksforGeeks

- Free Sprite Sheet Packer - CodeAndWeb:

Description: This is a free sprite sheet packer tool provided by CodeAndWeb. It helps in optimizing and packing individual sprite images into a single sprite sheet, which is efficient for game development.

Link: Free Sprite Sheet Packer – CodeAndWeb

- Image Splitter - Postcron:

Description: This is an online tool provided by Postcron for splitting images into multiple smaller images. It can be used to divide a large image into smaller sections or tiles for use in games or other applications.

Link: Image Splitter – Postcron

- Crop Image - Imgtools:

Description: This is an online image editing tool provided by Imgtools. It allows you to crop images by selecting a specific area, enabling you to extract or focus on specific parts of an image.

Link: Crop Image – Imgtools

- PicResize:

Description: PicResize is an online image editing tool that provides various image manipulation features. It allows you to resize, crop, rotate, and perform other editing operations on images.

Link: PicResize

- Aseprite:

Description: Aseprite is a pixel art and animation tool. It provides a user-friendly interface and features for creating pixel art graphics, animations, and sprite sheets for use in games or other projects.

Link: Aseprite

---*****---