INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HCMC

School of Computer Science and Engineering

Final Report - Group 24

Instructors: *Trần Thanh Tùng, Nguyễn Quang Phú*

Course: *Object-Oriented Programming*

TOPIC:
Game - Amazing Adventure

Members:
Châu Thịnh ITCSIU22240
Võ Gia Ân ITCSIU22241

CHAPTER I – INTRODUCTION

In the development of our game project, we conscientiously applied the theoretical foundations acquired through our Object-Oriented Programming (OOP) coursework, employing Java as the exclusive programming language. The principles of OOP guided our approach as we systematically organized our code into classes and objects, encapsulating various components of the game for enhanced modularity.

For the graphical aspects of the game display, we relied upon the Swing library, an integral component of Java's Abstract Window Toolkit (AWT). Swing furnished us with a comprehensive set of components, allowing the creation of a visually appealing and user-friendly interface encompassing windows, buttons, and other graphical elements essential for an immersive gaming experience.

Last but not least for the introduction, the game is an offline game, where players can play multiple times. The game can be played again whether the user wins or loses. The inventory, character’s state and the map will be reset whenever the Play Again button is chosen. Furthermore, users can pause the game while playing

CHAPTER II – RULES AND GAMEPLAY

How To Play

W: move up S: move down A: move left D: move right

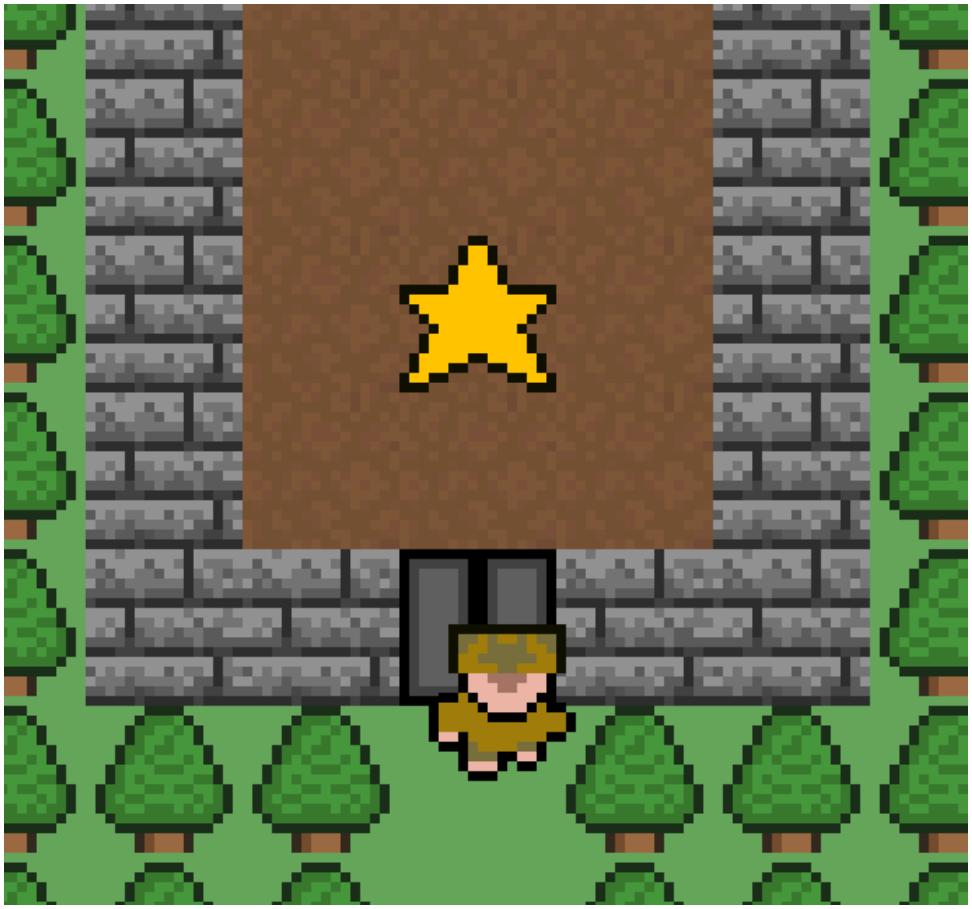
I: inventory P: Pause the game

Enter: melee attack (only enable when Player has got the Axe) / interact with NPC

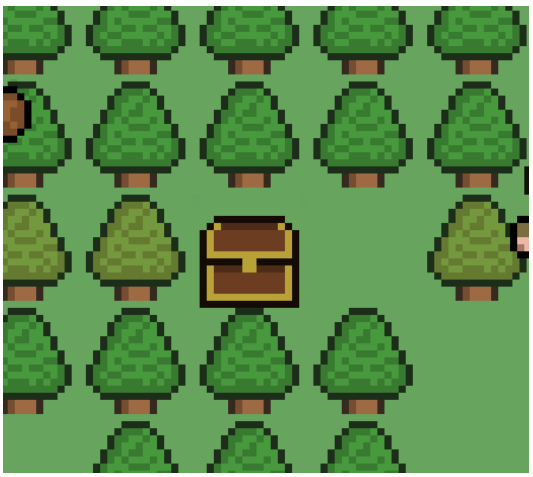
F: fire FireBall (only enable when Player has got the Red Potion)

Gameplay

This is a 2D adventuring game in which the player has to explore the entire map to reach the Star to win this game



To do that, the player needs to find keys to open chests, which may contain Buffs or an IronKey.



IronKey is used to open the Last Door to win.

Red Potion gives you power to shoot FireBall!

There are many monsters around so the player has to kill them to open the chest easily

If t

Player need to pick up a weapons(Axe) to kill monsters

Axe is also used to cut down the DryTree!

The Skeleton Boss won’t let Player reach the Star.
Beat it and win the game !

CHAPTER III – THE DETAILS OF THE GAME TECHNIQUE

3.1 Function of used class

- *Package AI:*
  - class Node:
    - The class is an essential component in the A* pathfinding algorithm. Each instance of the Node class represents a cell on a grid and contains attributes.
  - class PathFinder:
    - defines a PathFinder class implementing the A* (A-star) pathfinding algorithm. This algorithm efficiently computes the shortest path on a grid-based map, considering obstacles and

associated costs. The class utilizes a GamePanel and a 2D array of Node objects to represent the grid, with methods for initializing nodes, setting start and goal positions, calculating costs, and performing the search. The algorithm iteratively explores neighboring nodes, updating an open list, until the goal is reached or a maximum step limit is reached. The final path is then tracked and stored in a list.

- *Package entity*
  - class Entity:
    - Entity class represents all the entities (player, monster, npc, projectile, particle).
    - Storing the entities' information (life, damage, speed, direction, mana,...)
  - class NPC_Oldman:
    - setDialogue():
      - Set the String array of Quotes to talk to the player.
    - speak():
      - Display the Quotes on the screen.
    - getImage():
      - Load the image from files.
    - setAction():
      - Create a simple AI for the NPC to go in random directions.
  - class Player:
    - setDefaultValues():
      - Set the initial values for the player.
    - getPlayerImage():
      - Load the default images of the player from files.
    - getPlayerAttackImage():
      - Load the attacking images of the player from files.
    - draw():

        - Render the image of the player.
    - setDefaultPosition():
        - Set the initial position when the game starts.
    - damageMonster():
        - Attack the monster.
    - damageInteractiveTile():
        - Destroy the interactive tiles (cutting down dry trees).
    - interactNPC():
        - Talk with the NPC.
    - setItem():
        - Reset the items in the inventory when the game restarts.
    - pickupObject():
        - Pick up consumable objects and add them into the inventory.
- class Projectile:
    - set():
        - Set the initial shooting position, damage value, life(existing time).
    - update():
        - Update the position and check if the projectile collides with the target.
- class Particle:
    - draw():
        - Draw the particle.
    - update():
        - Display the motion.

- *Package main:*
    - class Main:
        - Use the JFrame from swing library to create a window.
        - Add the GamePanel to the window.
        - Start the game loop(game thread).

- class GamePanel:
  - Extend JPanel class
    - To draw the animation, display the map and sprites on the screen. (paintGameComponent(Graphics)).
  - Implement Runnable class:
    - We can use the class Thread to start the game loop (start())
    - The object whose run method is invoked when this thread is started. If null, this class's run method does nothing.
  - update():
    - Update the game in the realtime (position, interaction, events).
  - setupGame():
    - Setup the map, tiles, NPC, objects, monster,... before the game starts.
  - resetGame():
    - Set the game back to the initial condition to play a new game.
- class AssetSetter:
  - setObject():
    - Set the objects on the map.
  - setNPC():
    - Set the NPCs on the map.
  - setMonster():
    - Set the monster on the map.
  - clearMonster():
    - Clear the extending monster when reseting the game.
  - setInteractiveTile():
    - Set the interactive tiles.
- class CollisionChecker:
  - checkObject():
    - Check if the entity collides with objects.
  - checkTile():
    - Check whether the tile is solid or not.

    - checkEntity():
        - Used for player.
        - Check the type of the entity. If it is a monster, the player can damage it.
    - checkPlayer():
        - Used for monster.
        - Check if the monster collides with the player(for attacking player).
- class EventHandler:
    - hit():
        - Check if the player hit the special tiles.
    - checkEvent():
        - Check whether the event tiles can be touched or not.
    - damagePit():
        - If the player hit some pitting tree tiles, the player will be damaged.
    - healingPool():
        - If the player hits the pool, the life will be recovered.
    - resetEventDone():
        - If the player is more than 1 tile away from the touched tiles, it will reset and then can be touched again.
- class EventRect:
        - Extend the Rectangle class.
        - Represent the event Tile.
- class KeyHandler:
    - Implements KeyListener class.
    - Get the keyboard action then call the corresponding method.
    - keyPressed():
        - Check which key is pressed.
    - keyReleased():
        - Check which pressed key is released.
- class UI:
    - Used to display UI on the screen.

- showMessage():
  - Display the message.

- drawTitleScreen():
  - Display the title when the player picks up an object.

- drawPlayerMana():
  - Display the mana in the top left of the screen.

- drawPlayerLife():
  - Display the life in the top left of the screen(above the mana).

- drawInventory():
  - Display the inventory when the player opens it.

- drawEndScreen():
  - Display the Ending Screen when the player dies.

- drawWinScreen():
  - Display the Win Screen when the player wins.

- drawPauseScreen():
  - Display the Pause Screen when the player pauses.

- drawDialogueScreen():
  - Display dialogue when the player talks to NPC or picks up an Object.

- class UtilityTool:
  - The code defines a UtilityTool class in the "main" package, offering a method, scaledImage, to resize a given BufferedImage. This method takes the original image along

with the desired width and height, creating and returning a scaled version of the image.

- Package object:
    - *class Object:*
        - class serves as a base class for creating various objects within the game. Its functionality and characteristics can be extended or customized by creating subclasses that inherit from it.
    - *class Object_Axe:*
        - Initialize characteristics of Axe.
    - *class Object_Chest:*
        - Initialize characteristics of Chest.
    - *class Object_Door:*
        - Initialize characteristics of Door.
    - *class Object_Door_Iron:*
        - Initialize characteristics of Door Iron.
    - *class Object_FireBall:*
        - Initialize characteristics of FireBall.
    - *class Object_Heart:*
        - Initialize characteristics of Heart.
    - *class Object_Key:*
        - Initialize characteristics of Key.
    - *class Object_FinalKey:*
        - Initialize characteristics of FinalKey.
    - *class Object_Potion_Red:*
        - Initialize characteristics of Potion Red.
    - *class Object_Mana:*
        - Initialize characteristics of Mana.
    - *class Object_MetalPlate:*
        - Initialize characteristics of MetalPlate.
    - *class Object_Rock:*
        - Initialize characteristics of Rock.

    - *class Object_Star:*
        - Initialize characteristics of Star.

- *Package monster:*
    - class Monster_Orc:
        - setAction(): Determines the action of the orc, either following a path towards the player or making a random movement. It also checks whether the orc is attacking and locks its action for a certain duration.

        - damageReaction(): Handles the reaction of the orc when it takes damage, resetting the action lock counter and updating its direction based on the player's direction.
    - class Monster_Skeleton:
        - setAction(): Determines the action of the orc, either following a path towards the player or making a random movement. It also checks whether the orc is attacking and locks its action for a certain duration.

        - damageReaction(): Handles the reaction of the orc when it takes damage, resetting the action lock counter and updating its direction based on the player's direction.
- *Package tile_interactive:*
    - class InteractiveTile:
        - isCorrectItem(Entity entity): This method assesses whether the provided entity is the suitable item for interaction. Currently returning false, its extension relies on specific game logic implementation.

        - getDestroyedForm(): This method yields an InteractiveTile representing the tile's form after being destroyed. As of now, it returns null, with its eventual implementation contingent on the game's specifics.

- update(): This method dynamically updates the state of the interactive tile. In its present form, it manages invincibility logic, allowing the tile to become invulnerable for a specific duration.

- class IT_dryTree:
  - isCorrectItem(Entity entity): Overrides the method from the superclass, defaulting to true. This method's customization potential is grounded in game logic considerations.

  - getDestroyedForm(): Overrides the method from the superclass, returning an instance of IT_Trunk as the dry tree's destroyed form.

  - Various methods (getParticleColor(), getParticleSize(), getParticleSpeed(), getParticleMaxLife()): These methods furnish details about particle effects, encompassing attributes like color, size, speed, and maximum life.

  - draw(Graphics2D g2): Renders the dry tree on the screen, factoring in the player's position.

- class IT_Trunk:
  - The class inherits methods such as isCorrectItem, getDestroyedForm, and update from its superclass, InteractiveTile.

  - The draw(Graphics2D g2) method is not overridden, utilizing the implementation from the superclass.

- *Package tiles:*
    - class Tile:
        - Encapsulate the concept of a tile, present the Tile's image and collision detection.
    - class TileManager:
        - getTileImage(): Sets up various types of tiles with associated images and collision properties.
        - setup(int index, String imageName, boolean collision): Configures a specific tile at the given index with an image and collision property.
        - loadMap(String path): Loads the map from a specified text file, populating the mapTileNum array.
        - draw(Graphics2D g2): Renders the map on the screen, taking into account the player's position. Optionally, it can draw a path if the drawPath flag is set.

3.2 Main game flow

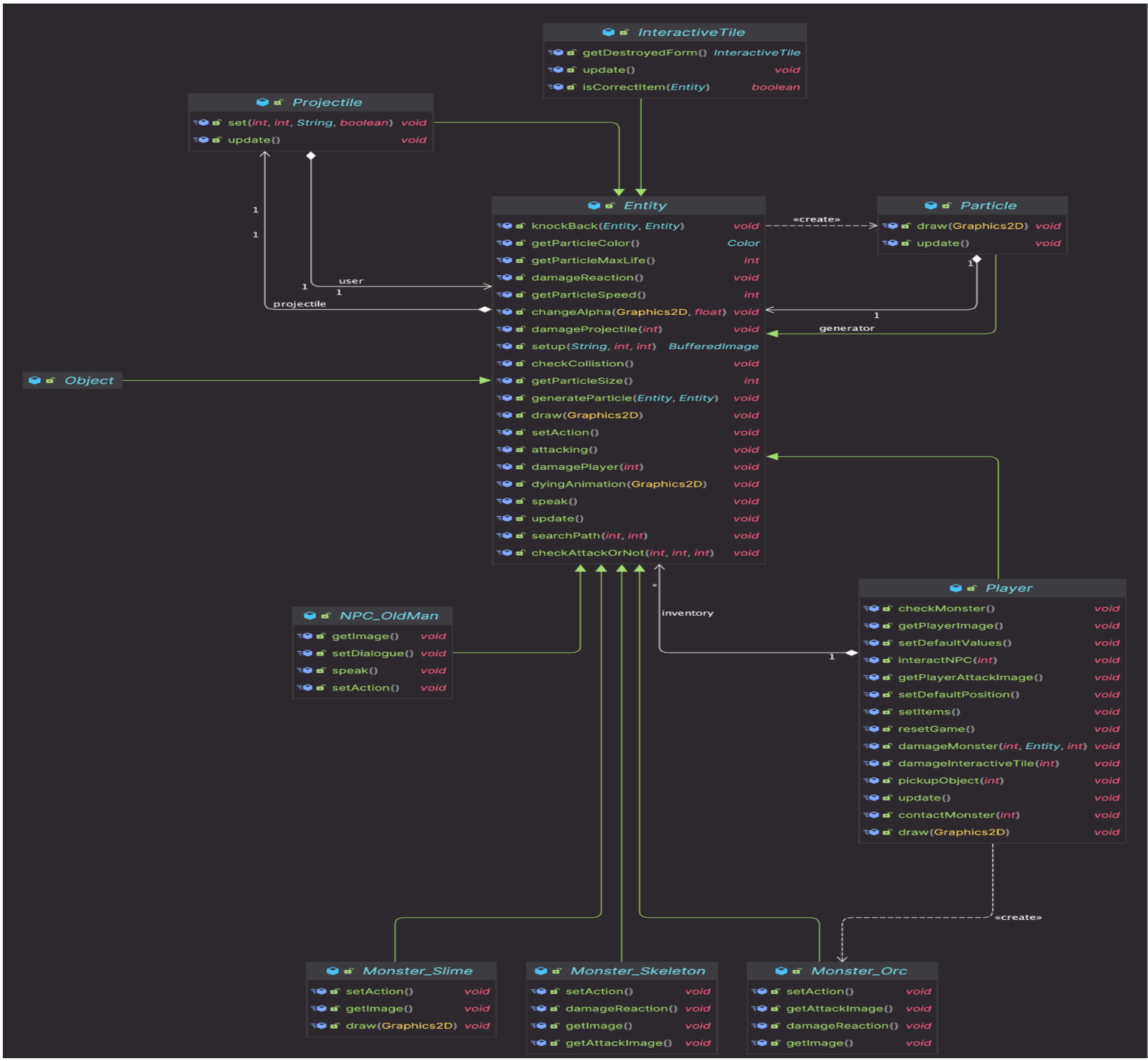
- First, class Main which contains the window, frame, game loop will:
    - Create a window by JFrame and add the GamePanel to it.
    - Call the setupGame() method of GamePanel to set up the game.
    - Call the startGameThread() method of GamePanel to start the game loop.
-
- Second, GamePanel will update game status by each frame using update():
    - Each frame will listen to the KeyListener and transmit it to KeyHandler to call the corresponding action.
        - Up, Down, Left, Right : move.
        - F: fire.
          Enter:
            - Talk with the NPC (if the target is an NPC).
            - Hit the monster (if the target is a monster).
        - P: pause the game.

    - I: open the player’s inventory.

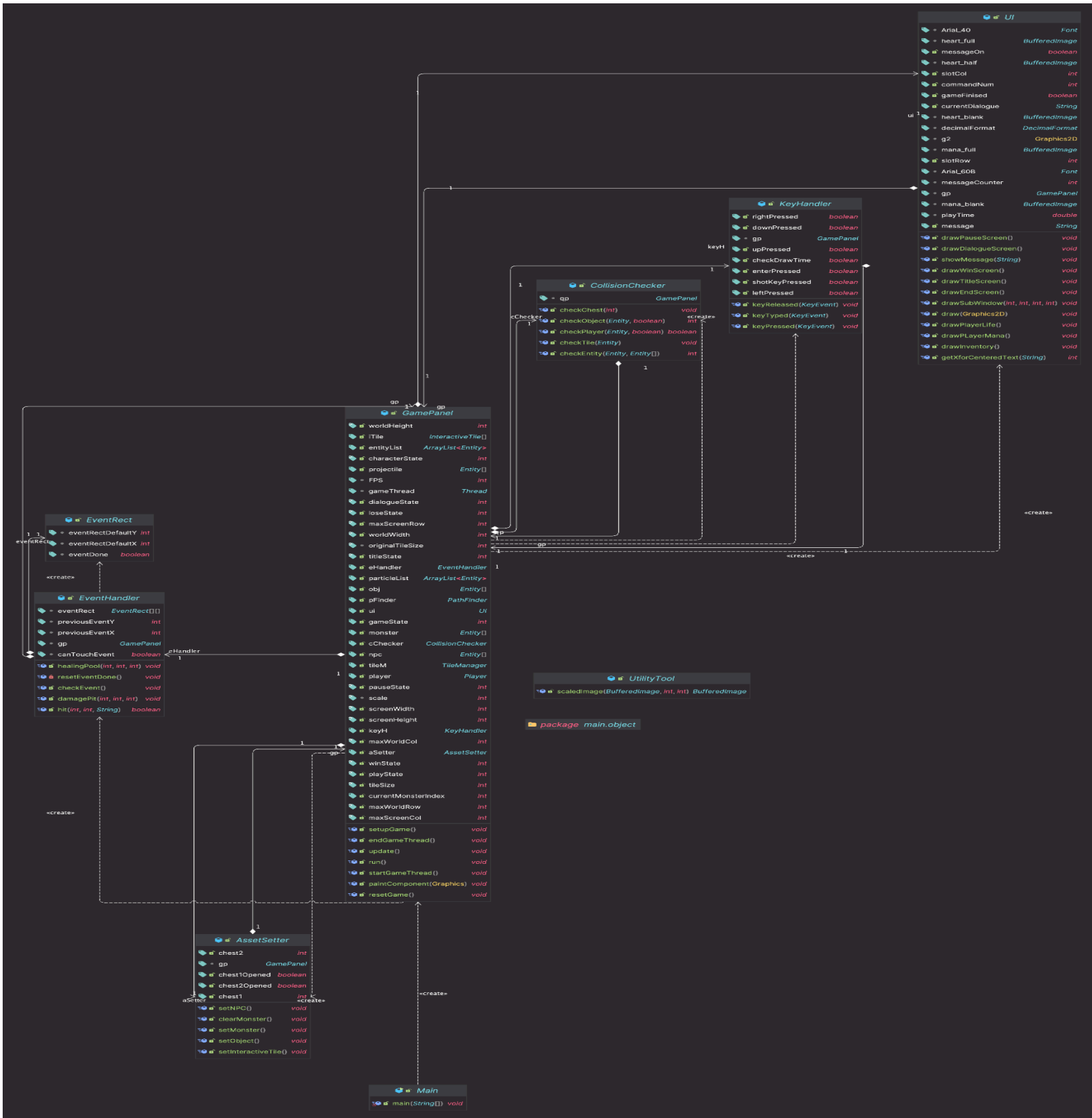
- The coordination between various classes shapes the dynamic game experience. Controllers, utility tools, entities, objects, monsters, tiles, and UI collaboratively contribute to the gameplay. Within the GamePanel's game loop, real-time updates unfold, ensuring fluid interactions between entities, tiles, and objects. Specialized classes manage player actions, events, and dialogue, collectively weaving an immersive gaming narrative.

CHAPTER IV – UML CLASS DIAGRAM

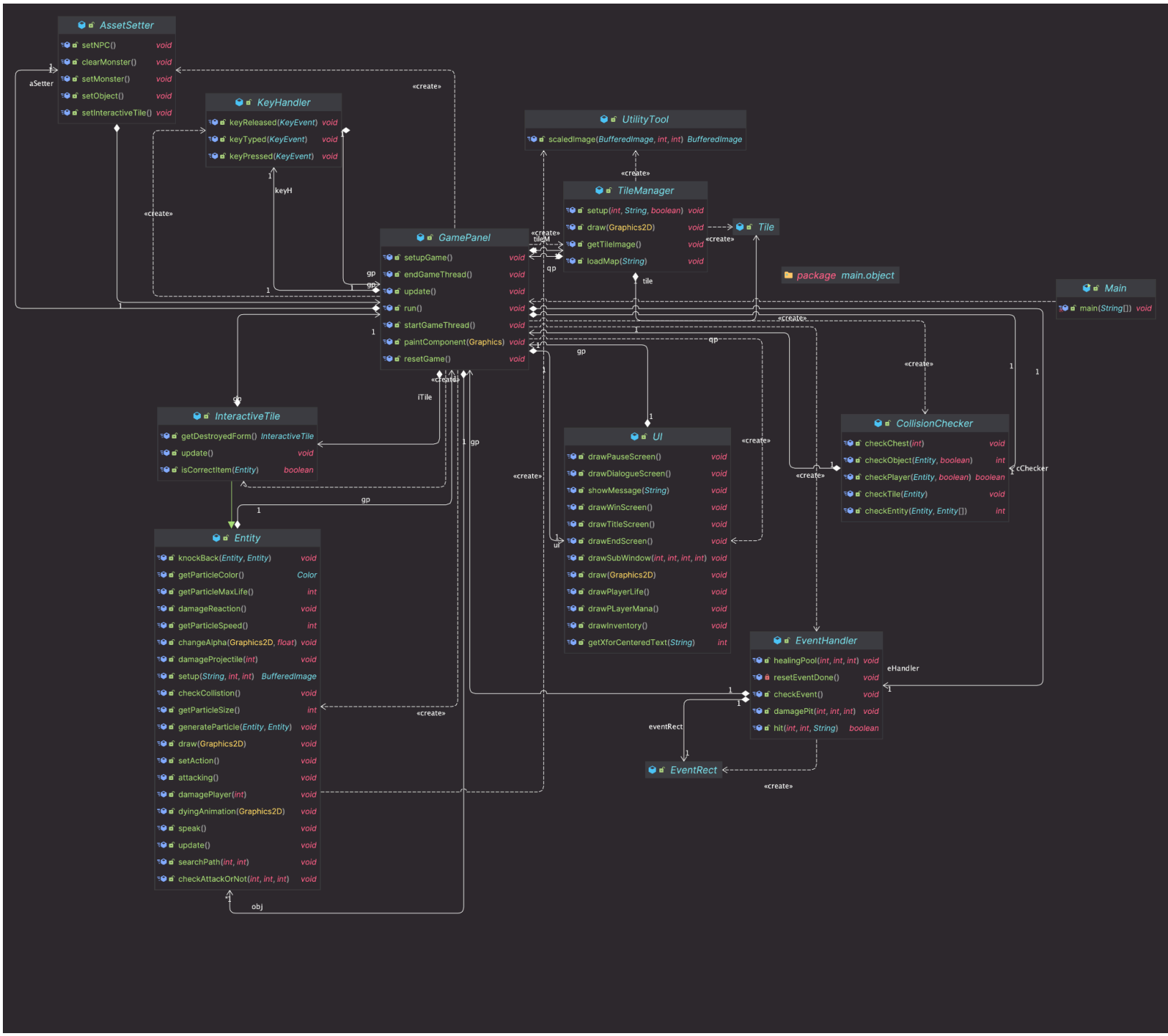
Entity UML:



Main UML:


UI
Arial_40 Font
heart_full BufferedImage
messageOn boolean
heart_half BufferedImage
slotCol int
commandNum int
gameFinised boolean
currentDialogue String
heart_blank BufferedImage
decimalFormat DecimalFormat
g2 Graphics2D
mana_full BufferedImage
slotRow int
Arial_60B Font
messageCounter int
gp GamePanel
mana_blank BufferedImage
playTime double
message String
drawPauseScreen() void
drawDialogueScreen() void
showMessage(String) void
drawWinScreen() void
drawTitleScreen() void
drawEndScreen() void
drawSubWindow(int, int, int, int) void
draw(Graphics2D) void
drawPlayerLife() void
drawPlayerMana() void
drawInventory() void
getXforCenteredText(String) int
KeyHandler
rightPressed boolean
downPressed boolean
gp GamePanel
upPressed boolean
checkDrawTime boolean
enterPressed boolean
shotKeyPressed boolean
leftPressed boolean
keyReleased(KeyEvent) void
keyTyped(KeyEvent) void
keyPressed(KeyEvent) void
CollisionChecker
gp GamePanel
checkChest(int) void
checkObject(Entity, boolean) int
checkPlayer(Entity, boolean) boolean
checkTile(Entity) void
checkEntity(Entity, Entity[]) int
GamePanel
worldHeight int
iTile InteractiveTile[]
entityList ArrayList<Entity>
characterState int
projectile Entity[]
FPS int
gameThread Thread
dialogueState int
loseState int
maxScreenRow int
worldWidth int
originalTileSize int
titleState int
eHandler EventHandler
particleList ArrayList<Entity>
obj Entity[]
pFinder PathFinder
ui UI
gameState int
monster Entity[]
cChecker CollisionChecker
npc Entity[]
tileM TileManager
player Player
pauseState int
scale int
screenWidth int
screenHeight int
keyH KeyHandler
maxWorldCol int
aSetter AssetSetter
winState int
playState int
tileSize int
currentMonsterIndex int
maxWorldRow int
maxScreenCol int
setupGame() void
endGameThread() void
update() void
run() void
startGameThread() void
paintComponent(Graphics) void
resetGame() void
EventRect
eventRectDefaultY int
eventRectDefaultX int
eventDone boolean
EventHandler
eventRect EventRect[][]
previousEventY int
previousEventX int
gp GamePanel
canTouchEvent boolean
healingPool(int, int, int) void
resetEventDone() void
checkEvent() void
damagePit(int, int, int) void
hit(int, int, String) boolean
UtilityTool
scaledImage(BufferedImage, int, int) BufferedImage
package main.object
AssetSetter
chest2 int
gp GamePanel
chest1Opened boolean
chest2Opened boolean
chest1 int
setNPC() void
clearMonster() void
setMonster() void
setObject() void
setInteractiveTile() void
Main
main(String[]) void
«create»
ui
keyH
cChecker
eventRect
eHandler
aSetter

General UML:


AssetSetter
setNPC() void
clearMonster() void
setMonster() void
setObject() void
setInteractiveTile() void
aSetter
KeyHandler
keyReleased(KeyEvent) void
keyTyped(KeyEvent) void
keyPressed(KeyEvent) void
keyH
UtilityTool
scaledImage(BufferedImage, int, int) BufferedImage
TileManager
setup(int, String, boolean) void
draw(Graphics2D) void
getTileImage() void
loadMap(String) void
Tile
tileM
tile
GamePanel
setupGame() void
endGameThread() void
update() void
run() void
startGameThread() void
paintComponent(Graphics) void
resetGame() void
gp
package main.object
Main
main(String[]) void
InteractiveTile
getDestroyedForm() InteractiveTile
update() void
isCorrectItem(Entity) boolean
iTile
UI
drawPauseScreen() void
drawDialogueScreen() void
showMessage(String) void
drawWinScreen() void
drawTitleScreen() void
drawEndScreen() void
drawSubWindow(int, int, int, int) void
draw(Graphics2D) void
drawPlayerLife() void
drawPLayerMana() void
drawInventory() void
getXforCenteredText(String) int
ui
CollisionChecker
checkChest(int) void
checkObject(Entity, boolean) int
checkPlayer(Entity, boolean) boolean
checkTile(Entity) void
checkEntity(Entity, Entity[]) int
cChecker
Entity
knockBack(Entity, Entity) void
getParticleColor() Color
getParticleMaxLife() int
damageReaction() void
getParticleSpeed() int
changeAlpha(Graphics2D, float) void
damageProjectile(int) void
setup(String, int, int) BufferedImage
checkCollistion() void
getParticleSize() int
generateParticle(Entity, Entity) void
draw(Graphics2D) void
setAction() void
attacking() void
damagePlayer(int) void
dyingAnimation(Graphics2D) void
speak() void
update() void
searchPath(int, int) void
checkAttackOrNot(int, int, int) void
obj
EventHandler
healingPool(int, int, int) void
resetEventDone() void
checkEvent() void
damagePit(int, int, int) void
hit(int, int, String) boolean
eHandler
eventRect
EventRect
«create»

CHAPTER V – EVALUATION

The execution of our game project was a tangible application of Object-Oriented Programming (OOP) principles using Java. This practical engagement enhanced our understanding of OOP concepts like encapsulation, inheritance, and polymorphism. The integration of the Swing library for graphical interface design not only improved the game's visual aesthetics but also offered hands-on experience in incorporating external libraries into Java applications.

Through this endeavor, we deepened our grasp of OOP by systematically organizing code into classes and objects. The collaborative dynamics among classes, spanning controllers, entities, and utility tools, underscored the importance of modularity and reusable code. The UML class diagram illustrated the intricate relationships, emphasizing the need for thoughtful class design and hierarchical clarity.

Moreover, the real-time functionality of the game loop within the GamePanel class demonstrated the practical implications of OOP in managing dynamic gameplay. Elements like collision detection mechanisms and specialized classes handling player actions and events showcased the effectiveness of OOP in creating a coherent and captivating gaming experience.

In summary, this project not only refined our programming proficiency but also provided a comprehensive application of OOP concepts, aligning seamlessly with the theoretical foundations learned in our Object-Oriented Programming coursework.

CHAPTER VI – REFERENCES

GitHub Tutorial

2D Game in Java

Java 2D Game Engine Development

Java games collision detection (zetcode.com)