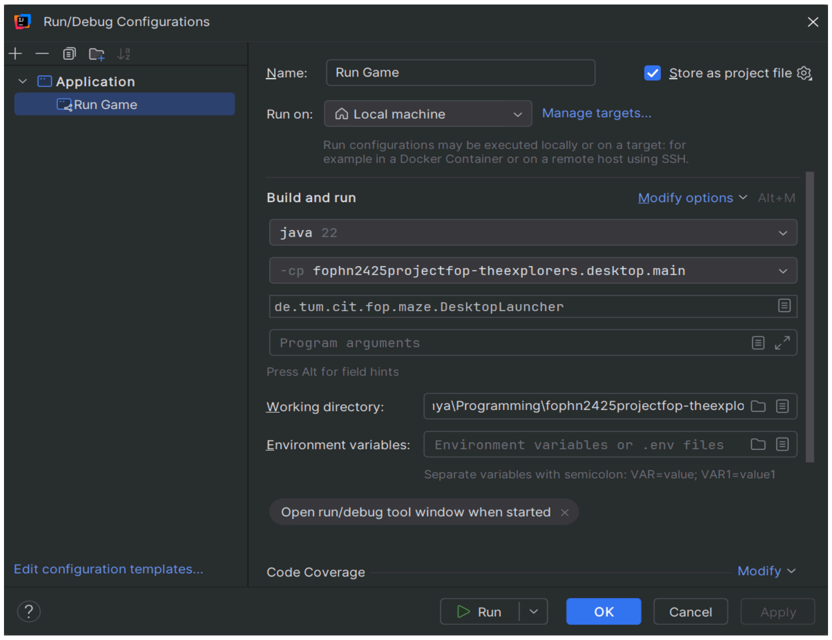
# README FILE: MAZE RUNNER GAME (theexplorers)

1. <u>Understanding Game mechanics</u>

The game is a tomb raider styled maze-runner game, where the purpose is to escape the maze by collecting all of the treasures, still protected by the ghost of the owner, before losing all lives . The game starts off with a menu for the user to choose a maze out of the 5 available. Upon selecting a maze, a player has to walk through the entrance and start collecting the treasures. The user can press escape anytime during the gameplay and the game will be paused. The user can then choose to resume or start afresh. Upon success, a victory screen is displayed with a score (the time taken to escape the maze). Upon failure, a Game Over screen is displayed indicating all the lives are lost. Throughout the entire gameplay, an arrow will be present near the character directing towards the exit at all times. Upon collision with the traps or the enemy, a sound effect is played with the camera zooming in on the player. An additional feature includes increasing speed by pressing the Shift key and the arrow key in the desired direction.

2. <u>How to run and use the Game</u>

To run the game, the IDE Intellij is recommended as this game was coded in there. Below is the setting required to run the game successfully:



Run/Debug Configurations

Application
Run Game

Name: Run Game
[✓ Store as project file]
Run on: Local machine
Manage targets...
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run
Modify options Alt+M

```
java 22
-cp fophn2425projectfop-theexplorers.desktop.main
de.tum.cit.fop.maze.DesktopLauncher
Program arguments
```

Press Alt for field hints

Working directory: ıya\Programming\fophn2425projectfop-theexplo

Environment variables: Environment variables or .env files

Separate variables with semicolon: VAR=value; VAR1=value1

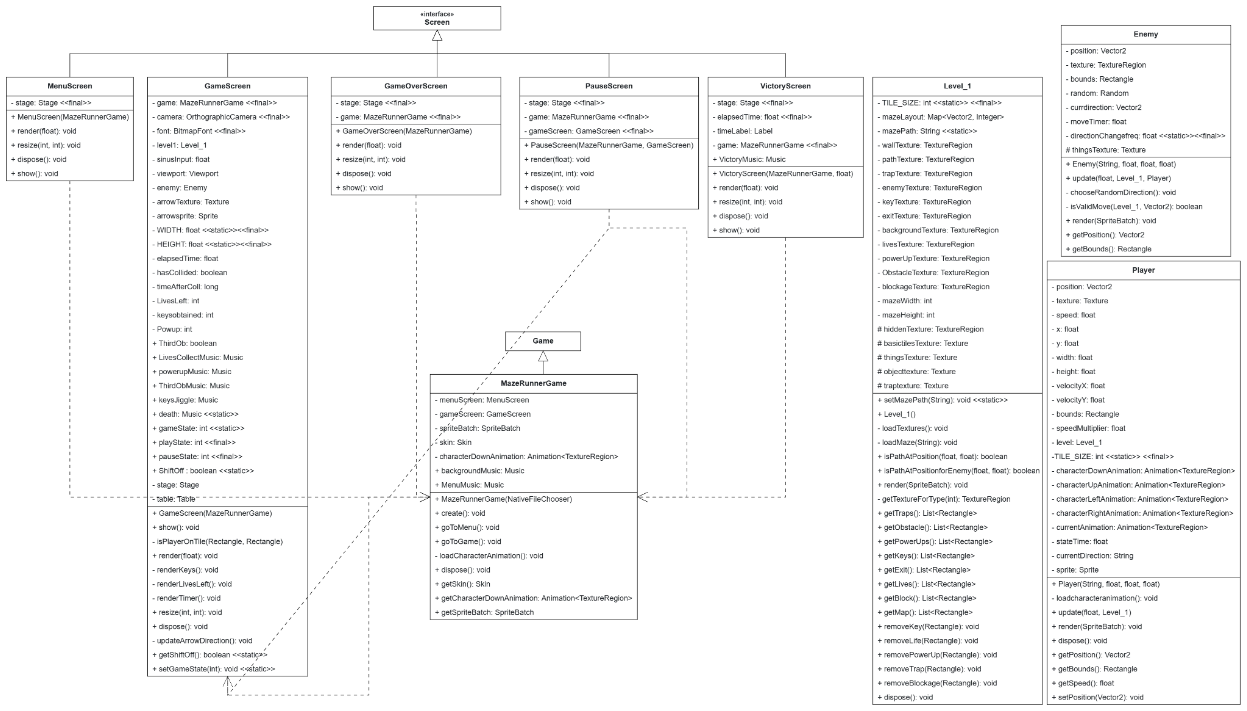
Open run/debug tool window when started

Code Coverage
Modify

Edit configuration templates...

Also, remove -XstartOnFirstThread when using Windows or Linux. All the necessary files and images used in the code are available under the assets section. Once the settings are sorted, simply run the code and the game should appear.

## 3. Class Hierarchy and code structure


«interface»
Screen
MenuScreen
- stage: Stage <<final>>
+ MenuScreen(MazeRunnerGame)
+ render(float): void
+ resize(int, int): void
+ dispose(): void
+ show(): void
GameScreen
- game: MazeRunnerGame <<final>>
- camera: OrthographicCamera <<final>>
- font: BitmapFont <<final>>
- level1: Level_1
- sinusInput: float
- viewport: Viewport
- enemy: Enemy
- arrowTexture: Texture
- arrowsprite: Sprite
- WIDTH: float <<static>><<final>>
- HEIGHT: float <<static>><<final>>
- elapsedTime: float
- hasCollided: boolean
- timeAfterColl: long
- LivesLeft: int
- keysobtained: int
- Powup: int
+ ThirdOb: boolean
+ LivesCollectMusic: Music
+ powerupMusic: Music
+ ThirdObMusic: Music
+ keysJiggle: Music
+ death: Music <<static>>
+ gameState: int <<static>>
+ playState: int <<final>>
+ pauseState: int <<final>>
+ ShiftOff : boolean <<static>>
- stage: Stage
- table: Table
+ GameScreen(MazeRunnerGame)
+ show(): void
- isPlayerOnTile(Rectangle, Rectangle)
+ render(float): void
- renderKeys(): void
- renderLivesLeft(): void
- renderTimer(): void
+ resize(int, int): void
+ dispose(): void
- updateArrowDirection(): void
+ getShiftOff(): boolean <<static>>
+ setGameState(int): void <<static>>
GameOverScreen
- stage: Stage <<final>>
- game: MazeRunnerGame <<final>>
+ GameOverScreen(MazeRunnerGame)
+ render(float): void
+ resize(int, int): void
+ dispose(): void
+ show(): void
PauseScreen
- stage: Stage <<final>>
- game: MazeRunnerGame <<final>>
- gameScreen: GameScreen <<final>>
+ PauseScreen(MazeRunnerGame, GameScreen)
+ render(float): void
+ resize(int, int): void
+ dispose(): void
+ show(): void
VictoryScreen
- stage: Stage <<final>>
- elapsedTime: float <<final>>
- timeLabel: Label
- game: MazeRunnerGame <<final>>
+ VictoryMusic: Music
+ VictoryScreen(MazeRunnerGame, float)
+ render(float): void
+ resize(int, int): void
+ dispose(): void
+ show(): void
Game
MazeRunnerGame
- menuScreen: MenuScreen
- gameScreen: GameScreen
- spriteBatch: SpriteBatch
- skin: Skin
- characterDownAnimation: Animation<TextureRegion>
+ backgroundMusic: Music
+ MenuMusic: Music
+ MazeRunnerGame(NativeFileChooser)
+ create(): void
+ goToMenu(): void
+ goToGame(): void
- loadCharacterAnimation(): void
+ dispose(): void
+ getSkin(): Skin
+ getCharacterDownAnimation: Animation<TextureRegion>
+ getSpriteBatch: SpriteBatch
Level_1
- TILE_SIZE: int <<static>> <<final>>
- mazeLayout: Map<Vector2, Integer>
- mazePath: String <<static>>
- wallTexture: TextureRegion
- pathTexture: TextureRegion
- trapTexture: TextureRegion
- enemyTexture: TextureRegion
- keyTexture: TextureRegion
- exitTexture: TextureRegion
- backgroundTexture: TextureRegion
- livesTexture: TextureRegion
- powerUpTexture: TextureRegion
- ObstacleTexture: TextureRegion
- blockageTexture: TextureRegion
- mazeWidth: int
- mazeHeight: int
# hiddenTexture: TextureRegion
# basictilesTexture: Texture
# thingsTexture: Texture
# objecttexture: Texture
# traptexture: Texture
+ setMazePath(String): void <<static>>
+ Level_1()
- loadTextures(): void
- loadMaze(String): void
+ isPathAtPosition(float, float): boolean
+ isPathAtPositionforEnemy(float, float): boolean
+ render(SpriteBatch): void
- getTextureForType(int): TextureRegion
+ getTraps(): List<Rectangle>
+ getObstacle(): List<Rectangle>
+ getPowerUps(): List<Rectangle>
+ getKeys(): List<Rectangle>
+ getExit(): List<Rectangle>
+ getLives(): List<Rectangle>
+ getBlock(): List<Rectangle>
+ getMap(): List<Rectangle>
+ removeKey(Rectangle): void
+ removeLife(Rectangle): void
+ removePowerUp(Rectangle): void
+ removeTrap(Rectangle): void
+ removeBlockage(Rectangle): void
+ dispose(): void
Enemy
- position: Vector2
- texture: TextureRegion
- bounds: Rectangle
- random: Random
- currdirection: Vector2
- moveTimer: float
- directionChangefreq: float <<static>><<final>>
# thingsTexture: Texture
+ Enemy(String, float, float, float)
+ update(float, Level_1, Player)
- chooseRandomDirection(): void
- isValidMove(Level_1, Vector2): boolean
+ render(SpriteBatch): void
+ getPosition(): Vector2
+ getBounds(): Rectangle
Player
- position: Vector2
- texture: Texture
- speed: float
- x: float
- y: float
- width: float
- height: float
- velocityX: float
- velocityY: float
- bounds: Rectangle
- speedMultiplier: float
- level: Level_1
-TILE_SIZE: int <<static>> <<final>>
- characterDownAnimation: Animation<TextureRegion>
- characterUpAnimation: Animation<TextureRegion>
- characterLeftAnimation: Animation<TextureRegion>
- characterRightAnimation: Animation<TextureRegion>
- currentAnimation: Animation<TextureRegion>
- stateTime: float
- currentDirection: String
- sprite: Sprite
+ Player(String, float, float, float)
- loadcharacteranimation(): void
+ update(float, Level_1)
+ render(SpriteBatch): void
+ dispose(): void
+ getPosition(): Vector2
+ getBounds(): Rectangle
+ getSpeed(): float
+ setPosition(Vector2): void

The UML diagram above shows all of the classes present in the code. There are a total of 9 classes. All of the screen classes: MenuScreen, GameScreen, GameOverScreen, PauseScreen, and VictoryScreen, implement the Screen Interface in LibGDX. MazeRunnerGame extends the Game class (also included in LibGDX). All of the screen classes are connected to MazeRunnerGame through dependency as they all use the MazeRunnerGame class in their respective constructors. The UML diagram above consists of arrows, the dashed arrows show dependency and the solid arrow pointing towards the interface shows inheritance. The GameScreen class has associations with the three other classes: Player, Enemy, and Level_1 to handle game logic and react to a scenario accordingly.