Architecture

ENG1 Team 16, ‘Team Team’

Luke Batten

Owen Crucifix

Samuel Humphreys

Robbie Parr

Jude Daniels-Smith

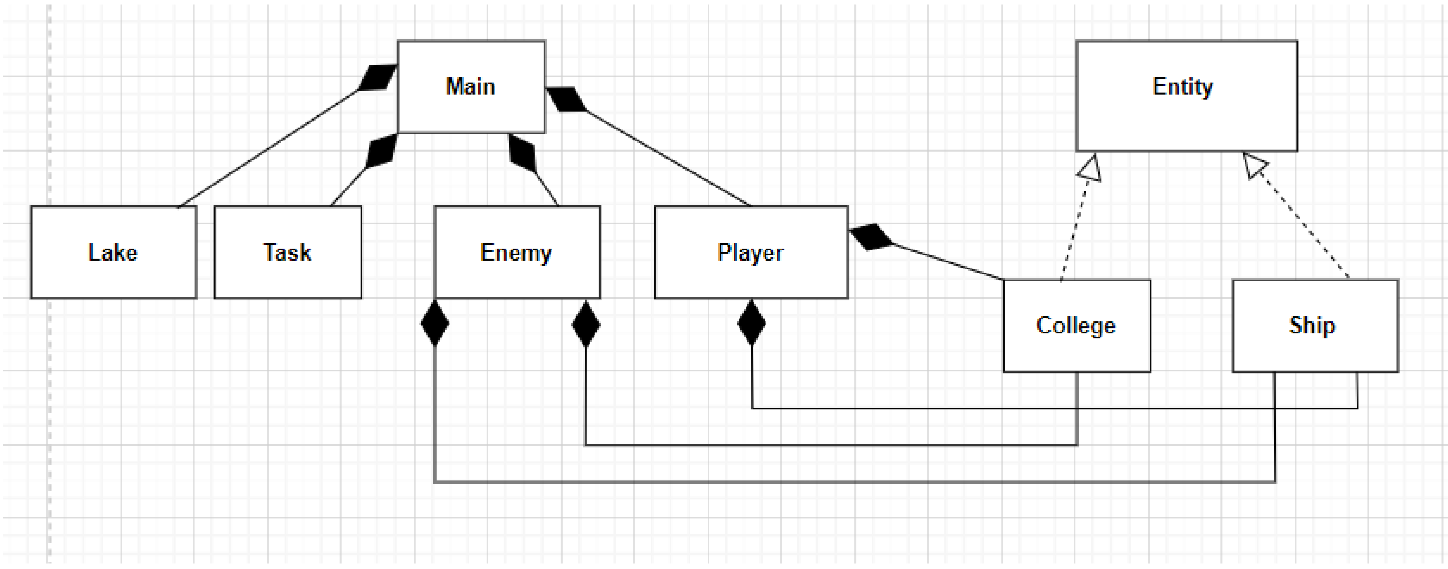
Alan Yang

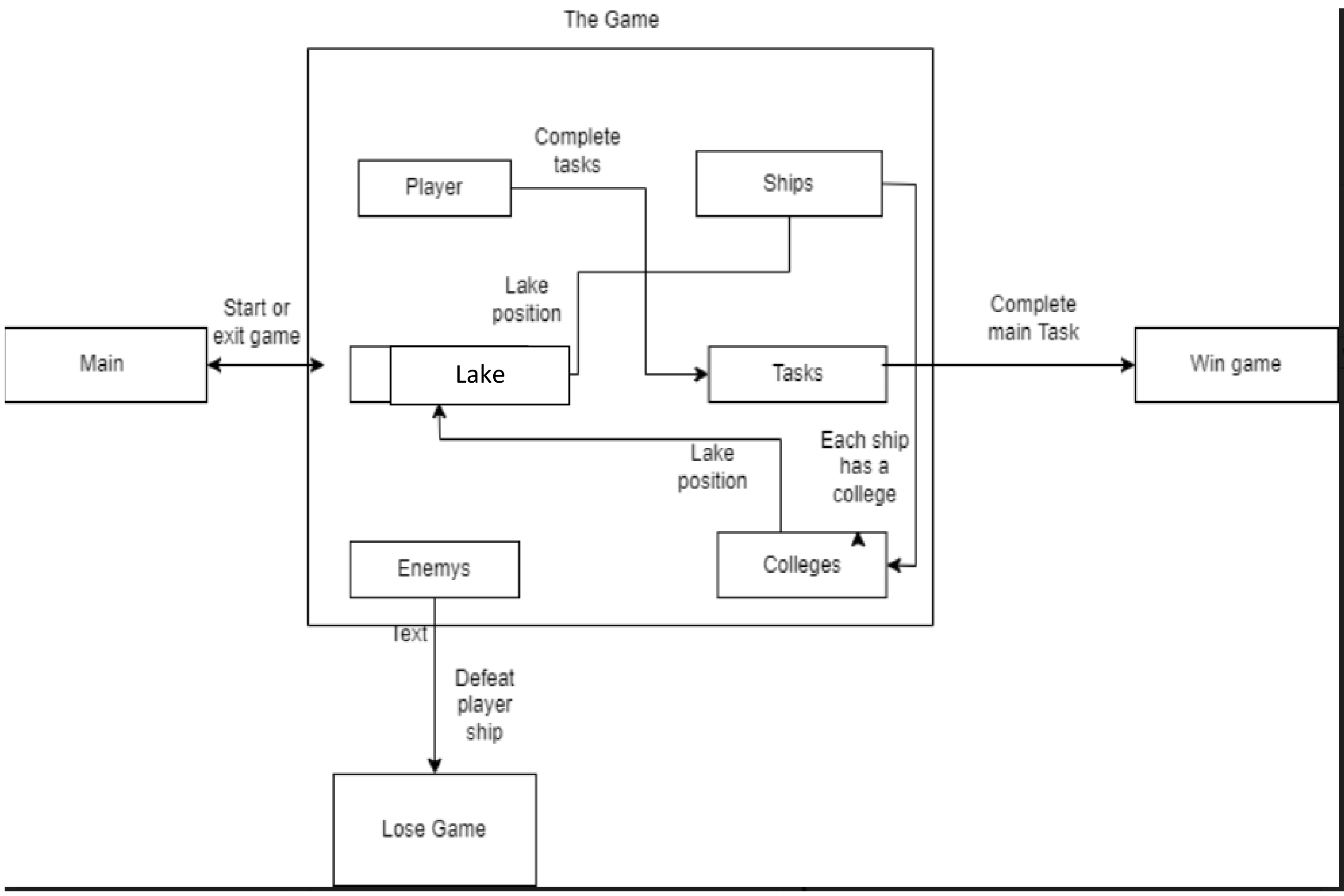
Architecture

This document aims to show how we plan to connect our requirements to the implementation of the software.

Abstract Architecture

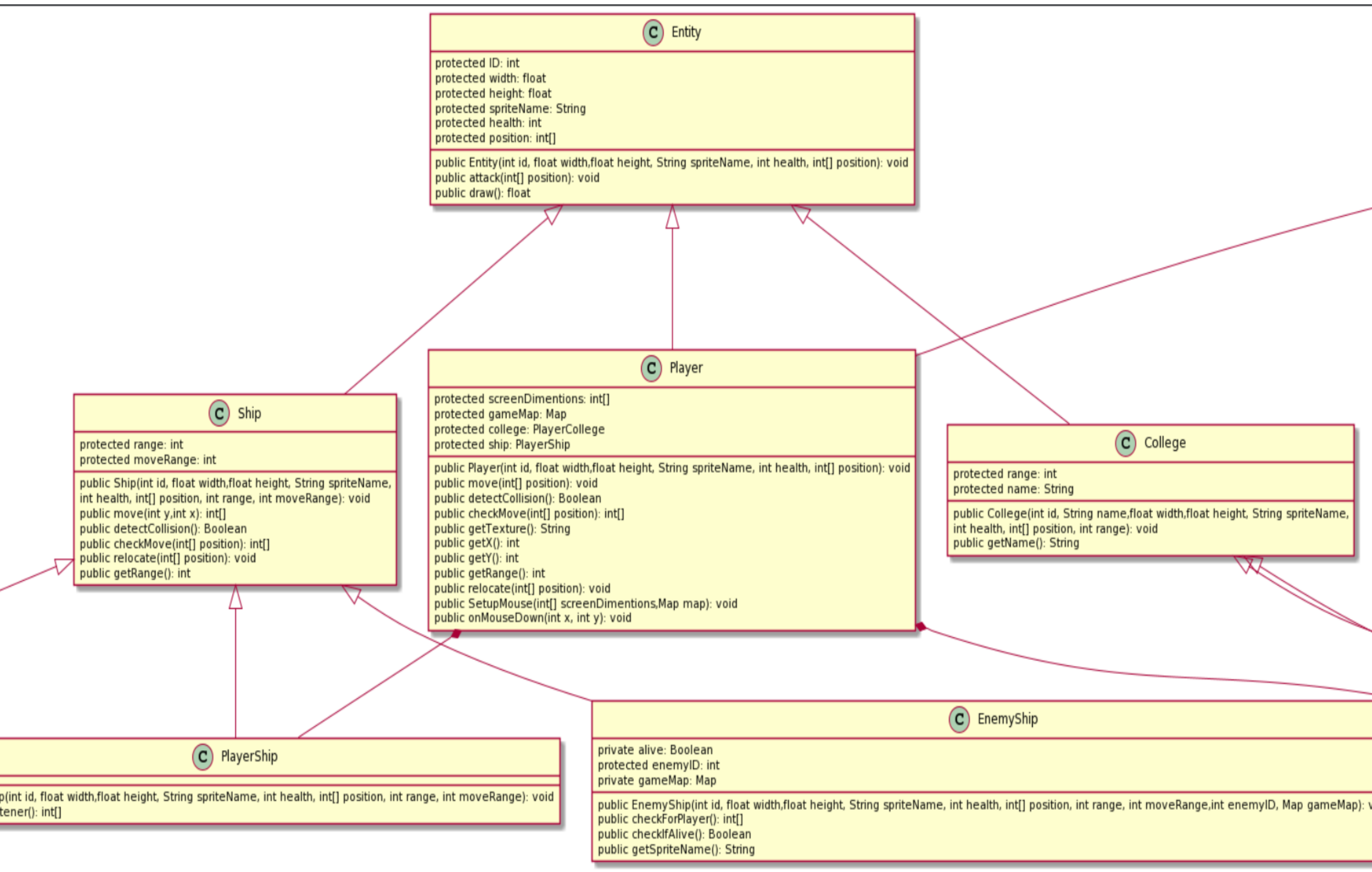
An abstraction of the architectural style that we plan to use. Both diagrams were created using draw.io.

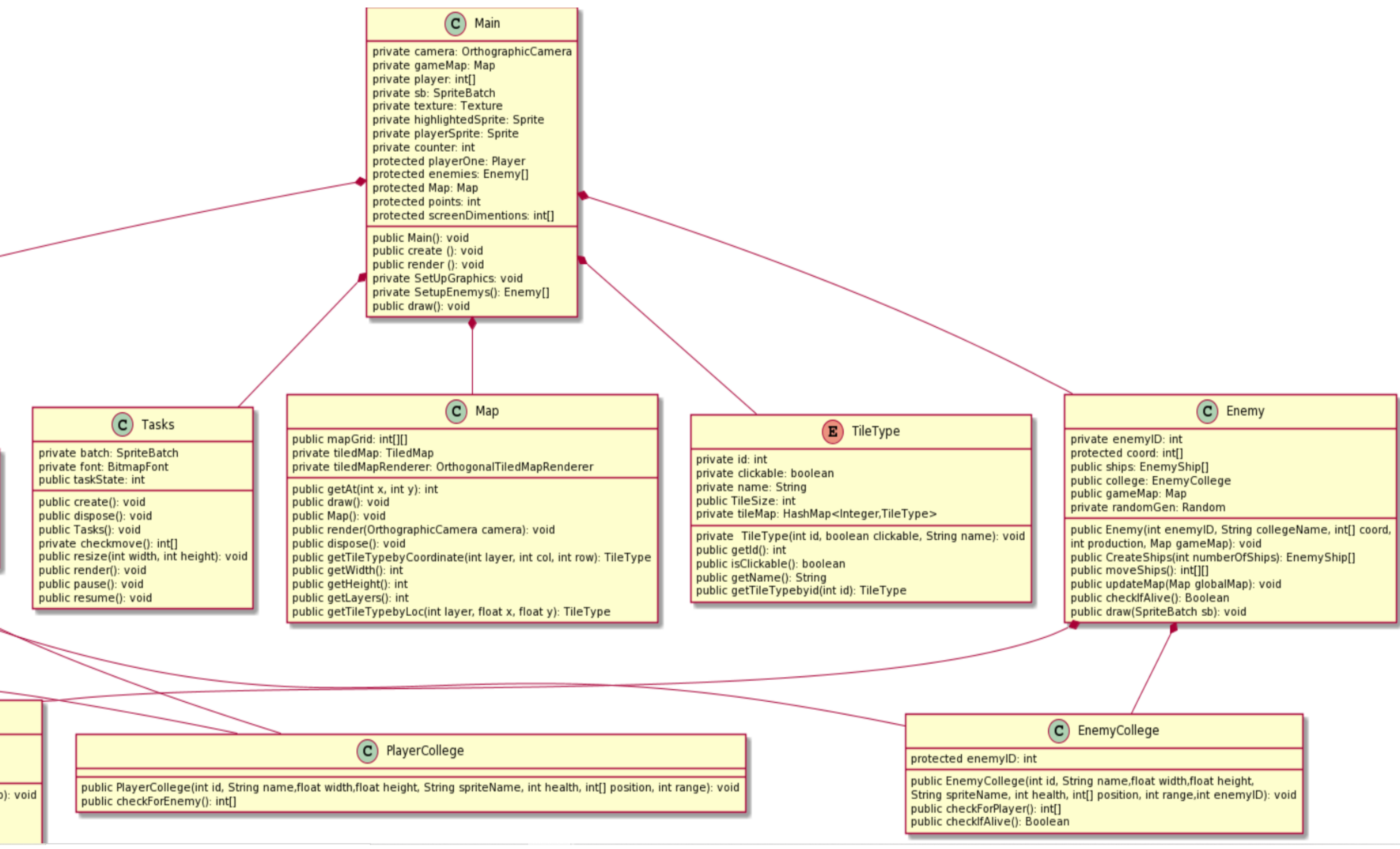




Concrete Architecture

Below is our concrete diagram which represents the implementation of the code. The diagrams will closely represent the code and design of the game implemented using libGDX. This was created using plantuml. For readability purposes, we cut the diagram into two


Entity
protected ID: int
protected width: float
protected height: float
protected spriteName: String
protected health: int
protected position: int[]
public Entity(int id, float width,float height, String spriteName, int health, int[] position): void
public attack(int[] position): void
public draw(): float
Ship
protected range: int
protected moveRange: int
public Ship(int id, float width,float height, String spriteName, int health, int[] position, int range, int moveRange): void
public move(int y,int x): int[]
public detectCollision(): Boolean
public checkMove(int[] position): int[]
public relocate(int[] position): void
public getRange(): int
Player
protected screenDimentions: int[]
protected gameMap: Map
protected college: PlayerCollege
protected ship: PlayerShip
public Player(int id, float width,float height, String spriteName, int health, int[] position): void
public move(int[] position): void
public detectCollision(): Boolean
public checkMove(int[] position): int[]
public getTexture(): String
public getX(): int
public getY(): int
public getRange(): int
public relocate(int[] position): void
public SetupMouse(int[] screenDimentions,Map map): void
public onMouseDown(int x, int y): void
College
protected range: int
protected name: String
public College(int id, String name,float width,float height, String spriteName, int health, int[] position, int range): void
public getName(): String
PlayerShip
p(int id, float width,float height, String spriteName, int health, int[] position, int range, int moveRange): void
tener(): int[]
EnemyShip
private alive: Boolean
protected enemyID: int
private gameMap: Map
public EnemyShip(int id, float width,float height, String spriteName, int health, int[] position, int range, int moveRange,int enemyID, Map gameMap): v
public checkForPlayer(): int[]
public checkIfAlive(): Boolean
public getSpriteName(): String


Main
private camera: OrthographicCamera
private gameMap: Map
private player: int[]
private sb: SpriteBatch
private texture: Texture
private highlightedSprite: Sprite
private playerSprite: Sprite
private counter: int
protected playerOne: Player
protected enemies: Enemy[]
protected Map: Map
protected points: int
protected screenDimentions: int[]
public Main(): void
public create (): void
public render (): void
private SetUpGraphics: void
private SetupEnemys(): Enemy[]
public draw(): void
Tasks
private batch: SpriteBatch
private font: BitmapFont
public taskState: int
public create(): void
public dispose(): void
public Tasks(): void
private checkmove(): int[]
public resize(int width, int height): void
public render(): void
public pause(): void
public resume(): void
Map
public mapGrid: int[][]
private tiledMap: TiledMap
private tiledMapRenderer: OrthogonalTiledMapRenderer
public getAt(int x, int y): int
public draw(): void
public Map(): void
public render(OrthographicCamera camera): void
public dispose(): void
public getTileTypebyCoordinate(int layer, int col, int row): TileType
public getWidth(): int
public getHeight(): int
public getLayers(): int
public getTileTypebyLoc(int layer, float x, float y): TileType
TileType
private id: int
private clickable: boolean
private name: String
public TileSize: int
private tileMap: HashMap<Integer,TileType>
private TileType(int id, boolean clickable, String name): void
public getId(): int
public isClickable(): boolean
public getName(): String
public getTileTypebyid(int id): TileType
Enemy
private enemyID: int
protected coord: int[]
public ships: EnemyShip[]
public college: EnemyCollege
public gameMap: Map
private randomGen: Random
public Enemy(int enemyID, String collegeName, int[] coord, int production, Map gameMap): void
public CreateShips(int numberOfShips): EnemyShip[]
public moveShips(): int[][]
public updateMap(Map globalMap): void
public checkIfAlive(): Boolean
public draw(SpriteBatch sb): void
PlayerCollege
public PlayerCollege(int id, String name,float width,float height, String spriteName, int health, int[] position, int range): void
public checkForEnemy(): int[]
EnemyCollege
protected enemyID: int
public EnemyCollege(int id, String name,float width,float height, String spriteName, int health, int[] position, int range,int enemyID): void
public checkForPlayer(): int[]
public checkIfAlive(): Boolean
): void

For readability as the diagram is across to pictures. Below state where lines go from the second picture to the first

- Playercollege class goes to Player and College class
- Enemycollege class goes to the college class.
- The line for Enemy class goes to Player class
- The line from Main class goes to Player class

Architecture Justification

We have used an inheritance approach within our concrete design firstly because we all have experience in designing software that way and also as relatively there are not that many entities an inheritance approach is appropriate.

Requirements to Abstract

- Main: UR_Game, The main state is used to start the game the user gets the college they will play with and then the game can begin.
- Win game: UR_Win, This is for when the user wins the game, The user has to complete a certain task for the game to be over so the game will end.
- Lose game: UR_Loss, When the user reacts to a stage where the game has been lost then it has to exit the game so it finishes and the user can't keep playing
- Lake: FR_Map, the lake represents the game map and everything apart of that for example the objects on the lake. Like ships and colleges
- Player: UR_Game, this represents the user entity and the objects that the user is in control of.
- Ship: UR_Ships, This is an object which will be on the map the user will control one and enemies will have their own. It inherits feacture from the entity.
- College: UR_Colleges, This is an object which will be on the map the user will control one and enemies each have their own. It inherits features from entity.
- Task: FR_Tasks, FR_Objective, This is a state where the user must complete tasks the be able to complete the final objective and then finally win the game
- Enemy: FR_NPC, This is an enemy entity that links to the enemy college and ships to interact with the user.

Abstract to Concrete

User: FR_Movement, NFT_inputLatency, FR_CollegeSelection, FR_Capture, FR_GainPoints, FR_GainPlunder

- Player is the Concrete version with all methods for the player to interact. It inherits from the entity superclass which allows players to move and act as an object in the program.
- The user is able to deal damage to colleges to complete certain tasks this is needed to meet the requirement FR_Capture Also the user acquires point while sailing over the lake

Enemies, FR_NPC

- Enemy is the concrete version with all methods for the Enemy ships and colleges. It is where the AI can be implemented and where the drawing of the Enemies is located. Allowing its inclusion/exclusion simple. This also enables it to be temporarily removed, which would help in testing other aspects of the project.

Ships, FR_Damage, FR_Health, FR_Speed

- Ship is a concrete version of all methods for the ships to use. PlayerShip, EnemyShip & FriendShip (cut off diagram) all inherit from Ship and it enables them to have positions, basic movement, a travelling range(moveRange), attack range (range) and health. Which (when combat is implemented) can be reduced to 0.

Mapping, FR_Graphics, NFT_Screen

- Map is the concrete version of the lake showing how it is loaded and spawned in. All entities interact with the map via the X and Y coordinate system provided by libgdx's orthographic camera. It consists of loading a .tmx file with the number of and location of a set of tiles, These tiles are held and within an enum called TileType. All tiles are rendered within the main method.

UI, NFT_Screen, FR_Graphics

- Although a work in progress at the time of writing this, the Tasks object, Map object & TileType object share the concrete version of all the methods related to these requirements. Map enables the user to see the map and processes most transactions related to it. TileType scales the map to fit the screen and enables each tile to be selected for movement, showing a red outline if you cannot move to it. And Tasks (in progress) will display the health and current task in the top left-hand corner.
- As a team, we made sure that the game was scaleable and worked on all our laptops to compile with NFT_Screen.