Architecture Representation

Abstract Architecture

The abstract representation aimed to help decide which areas of implementation would require the most work, and how to divide the team for this. The team met to discuss the relevant features the project required and created three overarching categories for development.

These categories are represented in the UML structure diagram [Figure 1].

- The GUI category covers how the game will be rendered and the interfaces the players will use to start and play the game.
- Rooms will cover the game's map and the ship's systems - including the win/loss decision.
- Entities cover the player's character movement and enemy AI.

The team also created a flow diagram [Figure 2] to represent the order of steps the game will take. This diagram will be useful during implementation to ensure classes are called in the correct order.

One of the client's requirements is that the game must be developed using the Java programming language. The Java language is optimised for Object Oriented Programming, which makes it easier to diagram how the game will work, using a class Diagram.

Languages & Tools

Due to the Object-oriented nature of this project, a UML class diagram is a suitable representation of the final architecture. Other UML diagrams are also suited to the project as they help the team understand how the project works. UML is also an industry standard for design representation [1], and so is a sensible language/tool to use.

The team used PlantUML [2], an open source application that uses textual descriptions to generate UML diagrams. This was both the official standalone application (GPLlicensed) and the Eclipse plugin from Hallvard Traetteberg (EPL-licensed).

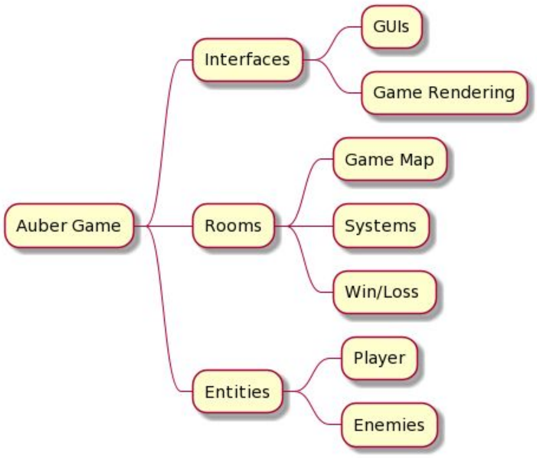


Figure 1: UML structure diagram for key development areas

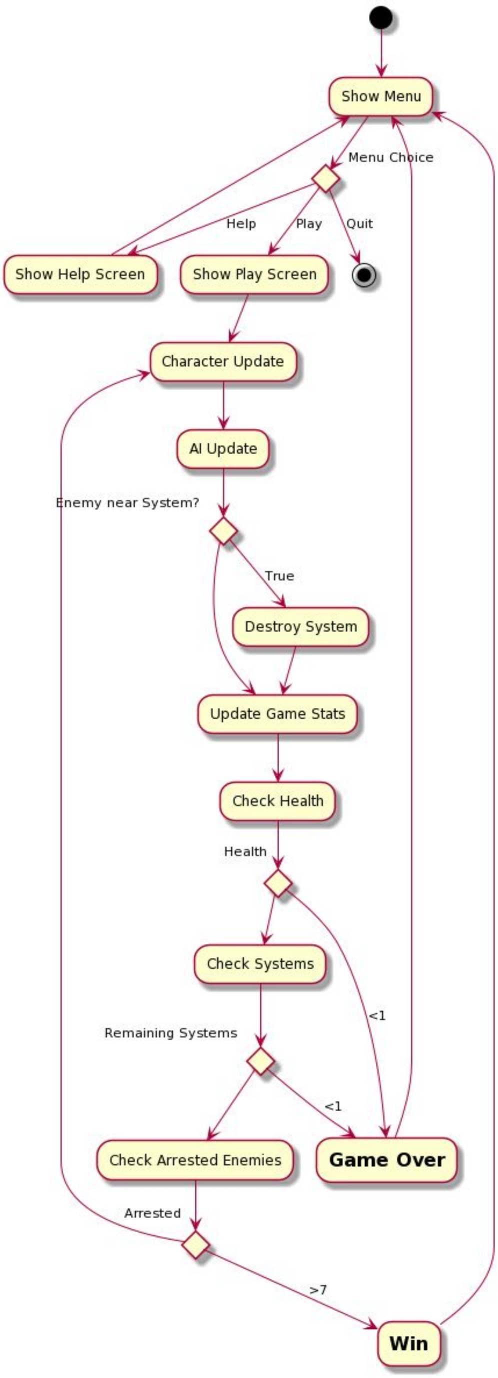
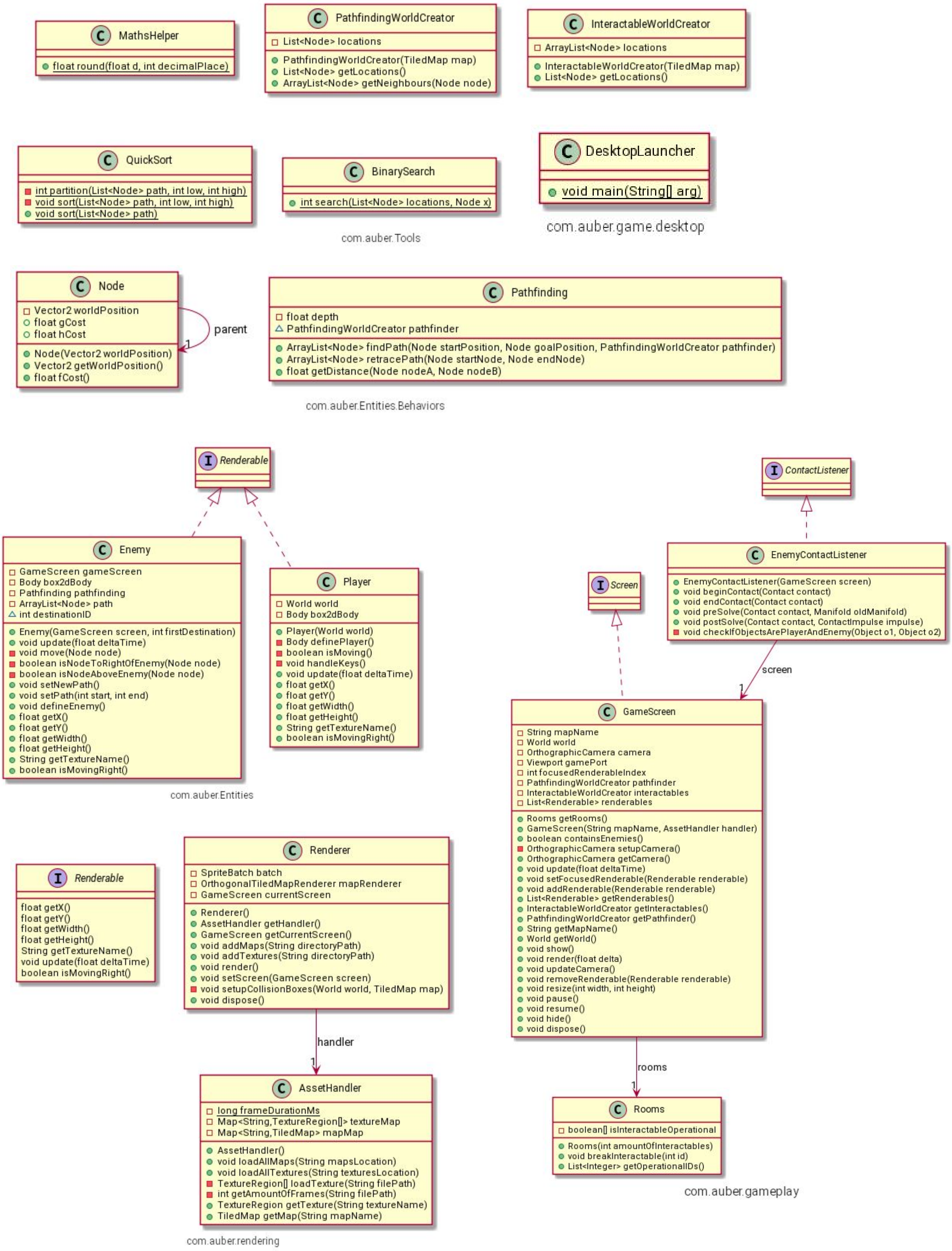


Figure 2: Flow Diagram showing how the game will run

Concrete Architecture

To convert our abstract architecture into a more concrete version, the classes have to be formally defined, using a UML class diagram. This diagram only consists of methods the team will create, and does not include those created by the library (libGDX) that is planned to be used to create the project.


MathsHelper
float round(float d, int decimalPlace)
PathfindingWorldCreator
List<Node> locations
PathfindingWorldCreator(TiledMap map)
List<Node> getLocations()
ArrayList<Node> getNeighbours(Node node)
InteractableWorldCreator
ArrayList<Node> locations
InteractableWorldCreator(TiledMap map)
List<Node> getLocations()
QuickSort
int partition(List<Node> path, int low, int high)
void sort(List<Node> path, int low, int high)
void sort(List<Node> path)
BinarySearch
int search(List<Node> locations, Node x)
com.auber.Tools
DesktopLauncher
void main(String[] arg)
com.auber.game.desktop
Node
Vector2 worldPosition
float gCost
float hCost
Node(Vector2 worldPosition)
Vector2 getWorldPosition()
float fCost()
parent
1
Pathfinding
float depth
PathfindingWorldCreator pathfinder
ArrayList<Node> findPath(Node startPosition, Node goalPosition, PathfindingWorldCreator pathfinder)
ArrayList<Node> retracePath(Node startNode, Node endNode)
float getDistance(Node nodeA, Node nodeB)
com.auber.Entities.Behaviors
Renderable
Enemy
GameScreen gameScreen
Body box2dBody
Pathfinding pathfinding
ArrayList<Node> path
int destinationID
Enemy(GameScreen screen, int firstDestination)
void update(float deltaTime)
void move(Node node)
boolean isNodeToRightOfEnemy(Node node)
boolean isNodeAboveEnemy(Node node)
void setNewPath()
void setPath(int start, int end)
void defineEnemy()
float getX()
float getY()
float getWidth()
float getHeight()
String getTextureName()
boolean isMovingRight()
Player
World world
Body box2dBody
Player(World world)
Body definePlayer()
boolean isMoving()
void handleKeys()
void update(float deltaTime)
float getX()
float getY()
float getWidth()
float getHeight()
String getTextureName()
boolean isMovingRight()
com.auber.Entities
ContactListener
EnemyContactListener
EnemyContactListener(GameScreen screen)
void beginContact(Contact contact)
void endContact(Contact contact)
void preSolve(Contact contact, Manifold oldManifold)
void postSolve(Contact contact, ContactImpulse impulse)
void checkIfObjectsArePlayerAndEnemy(Object o1, Object o2)
Screen
screen
1
GameScreen
String mapName
World world
OrthographicCamera camera
Viewport gamePort
int focusedRenderableIndex
PathfindingWorldCreator pathfinder
InteractableWorldCreator interactables
List<Renderable> renderables
Rooms getRooms()
GameScreen(String mapName, AssetHandler handler)
boolean containsEnemies()
OrthographicCamera setupCamera()
OrthographicCamera getCamera()
void update(float deltaTime)
void setFocusedRenderable(Renderable renderable)
void addRenderable(Renderable renderable)
List<Renderable> getRenderables()
InteractableWorldCreator getInteractables()
PathfindingWorldCreator getPathfinder()
String getMapName()
World getWorld()
void show()
void render(float delta)
void updateCamera()
void removeRenderable(Renderable renderable)
void resize(int width, int height)
void pause()
void resume()
void hide()
void dispose()
rooms
1
Rooms
boolean[] isInteractableOperational
Rooms(int amountOfInteractables)
void breakInteractable(int id)
List<Integer> getOperationalIDs()
com.auber.gameplay
Renderable
float getX()
float getY()
float getWidth()
float getHeight()
String getTextureName()
void update(float deltaTime)
boolean isMovingRight()
Renderer
SpriteBatch batch
OrthogonalTiledMapRenderer mapRenderer
GameScreen currentScreen
Renderer()
AssetHandler getHandler()
GameScreen getCurrentScreen()
void addMaps(String directoryPath)
void addTextures(String directoryPath)
void render()
void setScreen(GameScreen screen)
void setupCollisionBoxes(World world, TiledMap map)
void dispose()
handler
1
AssetHandler
long frameDurationMs
Map<String,TextureRegion[]> textureMap
Map<String,TiledMap> mapMap
AssetHandler()
void loadAllMaps(String mapsLocation)
void loadAllTextures(String texturesLocation)
TextureRegion[] loadTexture(String filePath)
int getAmountOfFrames(String filePath)
TextureRegion getTexture(String textureName)
TiledMap getMap(String mapName)
com.auber.rendering

Justification of Architectures

For our project, we chose to use an OOP paradigm, with different classes containing different functionalities of the game. This has allowed us to easily change different aspects of the game, and use inheritance to split functionality into subclasses and superclasses.

This approach also allows multiple members of the team to work on the code at once, as if the classes are planned out to have specific attributes and methods, different people can work on different classes and they will all function together without requiring intermediate functions to allow them to communicate.

For example, the class 'Enemy' (for the User Requirement UR_ENEMY) allows separate enemies to be created, and it means that the player can interact with them all individually. This allows us a more flexible design approach, and means that it is easy to create new entities within the game.
Also in the Entity package is the Player class (for the User Requirement UR_PLAYER) which defines the player's entity and how it interacts with the other entities in the game.

The Rendering package (com.auber.rendering) will be used by other classes to help render the game. By having a centralised rendering method, this doesn't have to be considered in the other methods.

The gamescreen class, along with the rooms class, fulfil user requirements UR_WORLD, UR_WORLD_ROOMS, UR_WORLD_INFIRMARY, UR_WORLD_SYSTEMS and UR_REALTIME.

Bibliography

[1] Unified Modelling Language, "What is UML?", [Online]. Available: https://www.uml.org/what-is-uml.htm
[2] PlantUML, "PlantUML", [Online]. Available: https://plantuml.com/