Architecture

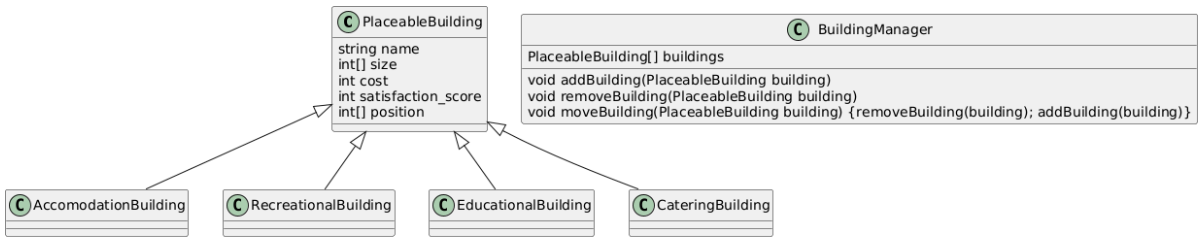
Group 9

Lewis Ramsey
Toby Rochester
Henry Sanger
Remi Shaw
Ethan Spiteri
William Timms
Antonio Tiron

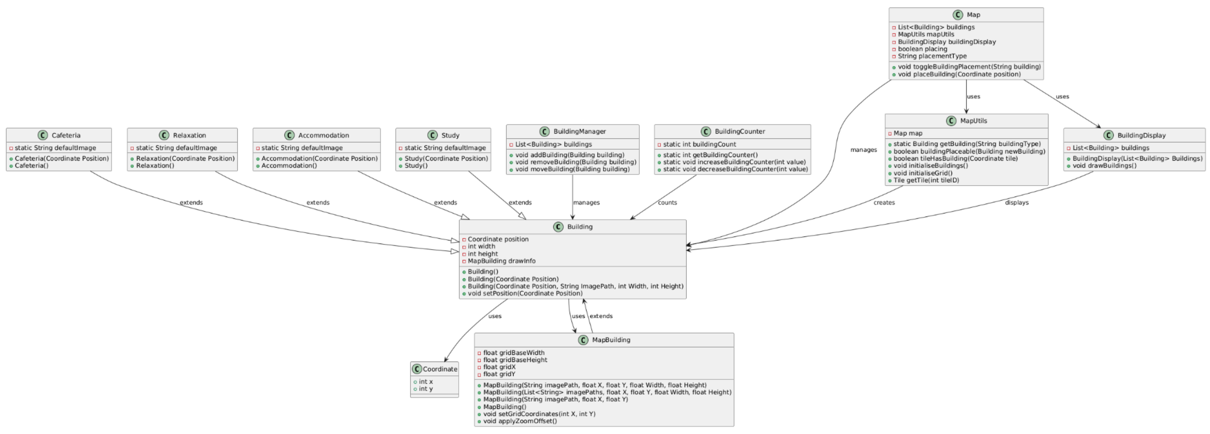
For reference:

The PlantUML extension was used in the Visual Studio Code interactive development environment to create the UML diagrams for the key parts of the architecture (for example map and buildings).

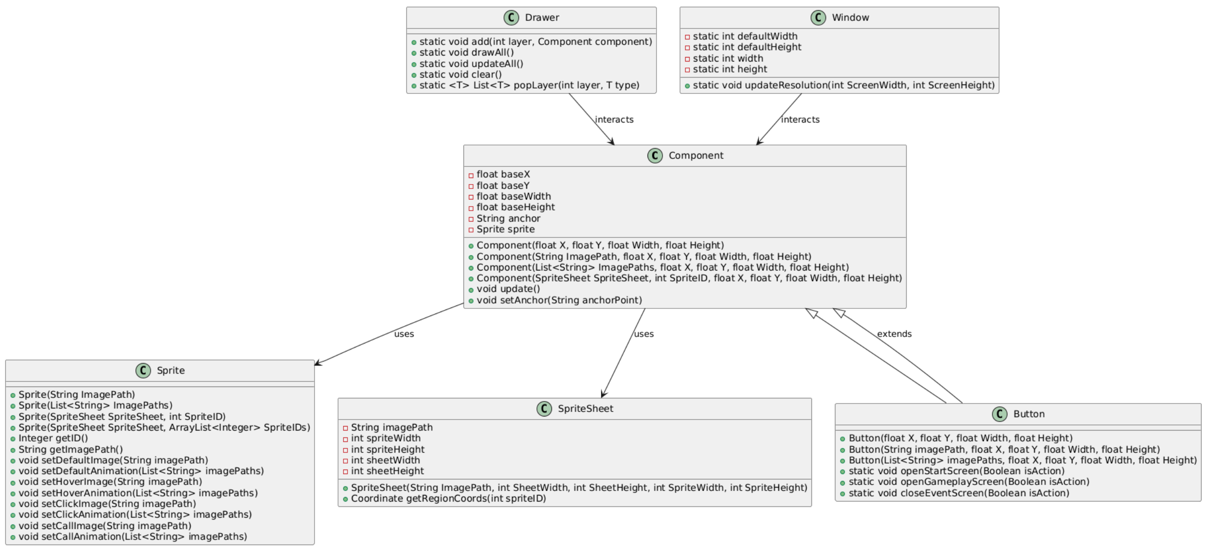
Initial building uml diagram (before code implementation)


PlaceableBuilding
string name
int[] size
int cost
int satisfaction_score
int[] position
BuildingManager
PlaceableBuilding[] buildings
void addBuilding(PlaceableBuilding building)
void removeBuilding(PlaceableBuilding building)
void moveBuilding(PlaceableBuilding building) {removeBuilding(building); addBuilding(building)}
AccomodationBuilding
RecreationalBuilding
EducationalBuilding
CateringBuilding

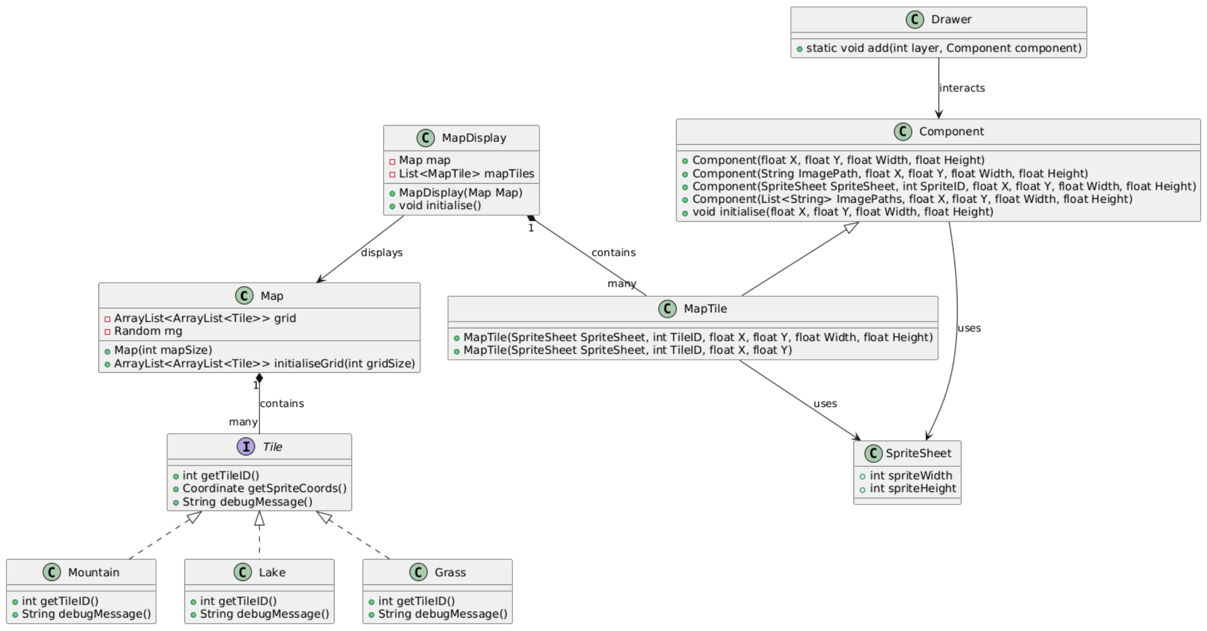
Final Building uml diagram (after code implementation)


Map
List<Building> buildings
MapUtils mapUtils
BuildingDisplay buildingDisplay
boolean placing
String placementType
void toggleBuildingPlacement(String building)
void placeBuilding(Coordinate position)
uses
uses
manages
MapUtils
Map map
static Building getBuilding(String buildingType)
boolean buildingPlaceable(Building newBuilding)
boolean tileHasBuilding(Coordinate tile)
void initialiseBuildings()
void initialiseGrid()
Tile getTile(int tileID)
BuildingDisplay
List<Building> buildings
BuildingDisplay(List<Building> Buildings)
void drawBuildings()
Cafeteria
static String defaultImage
Cafeteria(Coordinate Position)
Cafeteria()
Relaxation
static String defaultImage
Relaxation(Coordinate Position)
Relaxation()
Accommodation
static String defaultImage
Accommodation(Coordinate Position)
Accommodation()
Study
static String defaultImage
Study(Coordinate Position)
Study()
BuildingManager
List<Building> buildings
void addBuilding(Building building)
void removeBuilding(Building building)
void moveBuilding(Building building)
BuildingCounter
static int buildingCount
static int getBuildingCounter()
static void increaseBuildingCounter(int value)
static void decreaseBuildingCounter(int value)
extends
extends
extends
extends
manages
counts
creates
displays
Building
Coordinate position
int width
int height
MapBuilding drawInfo
Building()
Building(Coordinate Position)
Building(Coordinate Position, String ImagePath, int Width, int Height)
void setPosition(Coordinate Position)
uses
uses
extends
Coordinate
int x
int y
MapBuilding
float gridBaseWidth
float gridBaseHeight
float gridX
float gridY
MapBuilding(String imagePath, float X, float Y, float Width, float Height)
MapBuilding(List<String> imagePaths, float X, float Y, float Width, float Height)
MapBuilding(String imagePath, float X, float Y)
MapBuilding()
void setGridCoordinates(int X, int Y)
void applyZoomOffset()

Uml diagram for ui:


Drawer
static void add(int layer, Component component)
static void drawAll()
static void updateAll()
static void clear()
static <T> List<T> popLayer(int layer, T type)
Window
static int defaultWidth
static int defaultHeight
static int width
static int height
static void updateResolution(int ScreenWidth, int ScreenHeight)
interacts
interacts
Component
float baseX
float baseY
float baseWidth
float baseHeight
String anchor
Sprite sprite
Component(float X, float Y, float Width, float Height)
Component(String ImagePath, float X, float Y, float Width, float Height)
Component(List<String> ImagePaths, float X, float Y, float Width, float Height)
Component(SpriteSheet SpriteSheet, int SpriteID, float X, float Y, float Width, float Height)
void update()
void setAnchor(String anchorPoint)
uses
uses
extends
Sprite
Sprite(String ImagePath)
Sprite(List<String> ImagePaths)
Sprite(SpriteSheet SpriteSheet, int SpriteID)
Sprite(SpriteSheet SpriteSheet, ArrayList<Integer> SpriteIDs)
Integer getID()
String getImagePath()
void setDefaultImage(String imagePath)
void setDefaultAnimation(List<String> imagePaths)
void setHoverImage(String imagePath)
void setHoverAnimation(List<String> imagePaths)
void setClickImage(String imagePath)
void setClickAnimation(List<String> imagePaths)
void setCallImage(String imagePath)
void setCallAnimation(List<String> imagePaths)
SpriteSheet
String imagePath
int spriteWidth
int spriteHeight
int sheetWidth
int sheetHeight
SpriteSheet(String ImagePath, int SheetWidth, int SheetHeight, int SpriteWidth, int SpriteHeight)
Coordinate getRegionCoords(int spriteID)
Button
Button(float X, float Y, float Width, float Height)
Button(String imagePath, float X, float Y, float Width, float Height)
Button(List<String> imagePaths, float X, float Y, float Width, float Height)
static void openStartScreen(Boolean isAction)
static void openGameplayScreen(Boolean isAction)
static void closeEventScreen(Boolean isAction)

uml diagram for related map classes


Drawer
static void add(int layer, Component component)
interacts
Component
Component(float X, float Y, float Width, float Height)
Component(String ImagePath, float X, float Y, float Width, float Height)
Component(SpriteSheet SpriteSheet, int SpriteID, float X, float Y, float Width, float Height)
Component(List<String> ImagePaths, float X, float Y, float Width, float Height)
void initialise(float X, float Y, float Width, float Height)
MapDisplay
Map map
List<MapTile> mapTiles
MapDisplay(Map Map)
void initialise()
displays
1
contains
many
Map
ArrayList<ArrayList<Tile>> grid
Random rng
Map(int mapSize)
ArrayList<ArrayList<Tile>> initialiseGrid(int gridSize)
MapTile
MapTile(SpriteSheet SpriteSheet, int TileID, float X, float Y, float Width, float Height)
MapTile(SpriteSheet SpriteSheet, int TileID, float X, float Y)
uses
uses
SpriteSheet
int spriteWidth
int spriteHeight
1
contains
many
Tile
int getTileID()
Coordinate getSpriteCoords()
String debugMessage()
Mountain
int getTileID()
String debugMessage()
Lake
int getTileID()
String debugMessage()
Grass
int getTileID()
String debugMessage()

The architecture for buildings has made use of inheritance and classes to simplify the processes involved with buildings in the game. For example this design choice makes the placement and destruction of all types of buildings (user requirements: UR_PLACE_BUILD_SDY, UR_PLACE_BUILD_EAT, UR_PLACE_BUILD_HME, UR_PLACE_BUILD_RLX, UR_DESTROY_BUILD) much easier, because the main functionality of placing a generic building can be defined, then it can be called with specific parameters needed for the current building to be placed. In addition, the specific buildings inheriting from a general building class not only makes their design modular, but it also allows for editing/updates and maintenance to be carried out in a better way, due to changes to these building classes not affecting anything else as they are on the lowest level of the inheritance hierarchy.

In the case of the design for the map, this has been designed in a similar way to the buildings in terms of using inheritance in its implementation. This was done for easy updating by making the tile's classes modular. The functionalities for the map have been designed in classes and grouped based on their similarities and dependencies, which aids in understanding of code and allows for easier implementation of functionalities such as changing the size of the display, making the display automatically fit the window etc (user requirements: UR_SIZE_CHANGE).

As the code has been developed over the course of the project, the architecture has changed in all aspects; including the buildings, where currently the four types of buildings now inherit from a class Building, which has been done this way to reduce code duplication. In addition the class building manager manages the placement of buildings, including functions such as addBuilding(), removeBuilding() and moveBuilding() and it manages the building function. The class BuildingCounter counts the class Building therefore counting every building placed or removed (incrementing and decrementing respectively) which relates to the functional system requirement FR_BUILDING_COUNTER. These cover the user requirements similar to the initial design which are UR_PLACE_BUILD_SDY, UR_PLACE_BUILD_EAT, UR_PLACE_BUILD_HME, UR_PLACE_BUILD_RLX and UR_DESTROY_BUILD. These changes to the architecture of buildings were made to make the functionalities of placing buildings more efficient and to be able to implement the process of counting buildings, relating to scoring the game.

The architecture of the map has also had a large redesign, where the Map class now uses two other classes, MapUtils and BuildingDisplay, where MapUtils manages class Building and initialises (creates) the buildings to be placed, and BuildingDisplay displays the buildings.In terms of the buildings being placed on specific locations on the map, relating to user requirements UR_PLACE_BUILD_SDY, UR_PLACE_BUILD_EAT, UR_PLACE_BUILD_HME and UR_PLACE_BUILD_RLX; the class Building uses class Coordinate to place buildings on specific locations on the map, also using the class MapBuilding to further this functionality. These changes were made to implement more precise placement of buildings in a more modular way, to improve code quality.

In regards to the user interface and how that interacts with the user, more functionalities have been added to the window class that work outside of the users domain, such as updating the resolution when the size of the window is changed. In addition to this the functionality of buttons has been added to the system, where this change allows for more intuitive interaction between the user and the system, relating to the user requirement UR_UX. The button class extends the component class which is used by the window and draw class, and the component class also uses the Sprite and Spritesheet class. This change in architecture to the usability of the system drastically improves the users experience when playing the game, as more intuitive interaction means less complex thought processes and less problems to come across for the user. This relates to the nonfunctional system requirement NFR_OPERABILITY, in regards to making the game accessible to users with a range of computer knowledge.