UNIVERSITY OF MALTA
FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

CPS3231 Assignment : BiniTool

Keith Bugeja, Sandro Spina

Instructions:

1. This is an **individual** assignment; the implementation carries **60**% of the final CPS3231 grade, while the report together with a recorded video presentation carry the remaining **40**%.
2. The report and all related files (including code) must be uploaded to the VLE by the indicated deadline. Before uploading, archive all files into a compressed format such as ZIP. It is your responsibility to ensure that the uploaded file and all contents are valid.
3. The video presentation should not exceed ten minutes in length. It shall be hosted on your university Google Drive and a link included in the report. In the presentation you will discuss your **BiniTool** implementation while providing a demonstration.
4. Everything you submit must be your original work. All source code will be run through plagiarism detection software. Please read carefully the plagiarism guidelines on the ICT website.
5. Reports (and code) that are difficult to follow due to poor writing-style, organisation or presentation will be penalised.
6. Each individual may be asked to discuss their implementation with the examiners, at which time the program will be executed and the design explained. The outcome may affect the final marking.

Introduction

Computer-aided design (CAD) is the use of computers to aid in the creation and visualisation, amongst others, of a design. CAD finds wide use in applications such as prosthetic, industrial and architectural design, and has thus been a major driving force for research in computer graphics.

The goal of this assignment is to implement the small architectural design tool, described in this document: **BiniTool**. This assignment will be evaluated on a number of criteria, including correctness, structure, style and documentation. Therefore, your code should do what it purports to do, be organised into functions and modules, be well-indented, well-commented and descriptive (no cryptic variable and function names), and include a design document describing your implementation in detail.

Deliverables

Upload your source code, including any unit tests and additional utilities, through VLE by the specified deadline. Make sure that your code is properly commented and easily readable. Be careful with naming conventions and visual formatting. Include a report describing:

- the design of your system, including any structure, behaviour and interaction diagrams which might help;
- any implemented mechanisms which you think deserve special mention;
- your approach to testing the system, listing test cases and results;
- a list of bugs and issues of your implementation.

Do **not** include any code listing in your report; only snippets are allowed.

Record a video presentation of not more than ten minutes where you showcase and discuss your **BiniTool** implementation. Alongside the demonstration, you are free to show snippets of code, discuss shortcomings and features, bugs and accomplishments. Upload the video presentation to your Google Drive and link to it from your report.

BiniTool

In this assignment, you will develop **BiniTool**, an architectural design tool used to construct and walk through building floor plans. Your implementation should support the creation of arbitrarily sized rooms, connected through doorways and populated with various pieces of furniture and other objects. The tool will operate in two modes, a **design view**, where the user constructs the floor plans, designing the rooms and choosing materials, doorways, windows and placing furniture. This mode operates through an overhead view on the floor plan. A second mode, dubbed **exploration view**, allows a user to explore the created floor plan. This mode uses higher fidelity graphics and supports both a first-person walkthrough mode with proper collision detection and a free-roam disembodied mode, which allows the camera to hover and pass through walls and objects. Exploration view uses skyboxes to simulate different times during the day.

Design View

In design view, the user can create new floor plans and populate them with furniture, fixtures and other decorations. This mode triggers an orthographic overhead view of the floor plan, where the user can pan (translate camera along x-z plane) and zoom in or out. Initially, when the tool is started, the floor is empty and the camera is centred at $(0, 5, 0)$, with a forward vector of $(0, -1, 0)$. At this stage, the user may opt to load an existing floor plan or create a new one.

Task 1

(a) Create a user interface (UI) for design view with an overhead orthographic camera that lets the user pan and zoom.

   i) Camera pans along the x-z plane using the WASD or cursor keys on the keyboard.

   ii) Camera zooms in and out using the + and - keys respectively.

[5 marks]

Room Creation

Designs in **BiniTool** are room-centric, and a new design starts with the creation of the first room, which is treated as a special case: the user need only specify the width and breadth of the room and the tool will create it, centred on the x-z plane at $(0, 0)$. The y-coordinate of the room is such that the floor plane is oriented towards positive-y, with a surface normal of $(0, 1, 0)$. Note that for simplicity's sake we assume rooms to be rectangular. For any subsequently created rooms, the user has to choose an anchor room and a wall in addition to the dimensions of the room. The tool creates the new room adjacent to the anchor, sharing the selected wall. Newly created rooms need to be assigned materials for the floor, the walls and the ceiling.

Task 2

(a) Extend the design view to allow creation of new rooms.

i) Rooms are rectangular, with user-specified length, width and materials.

ii) First room is centred at $(0, 0)$; subsequent ones are created adjacent to an existing room.

[10 marks]

Doorways and Windows

The user may choose to place doorways between two adjacent rooms, to link them together and make one directly accessible from the other. Doorway-width is user-specified. Windows may also be added to rooms; they can be added to walls that belong to one room: i.e., the wall must not be shared by two adjacent rooms.

Task 3

(a) Extend the design view to allow for the creation of doorways and windows.

i) A single (primary) doorway representing the main entrance that is on a wall that is not shared between rooms.

ii) Secondary doorways are created between rooms sharing the same wall.

iii) Windows are created on walls that are not shared between rooms.

iv) Doorways and windows should not overlap one another.

v) Doorway and window width is user-specified.

vi) A path should exist between any two rooms in the floor plan.

[15 marks]

Editing Rooms in Design View

The user should be able to change the size and materials of an existing room. Rooms can be moved as long as they do not end up forming islands or overlapping other rooms. Doorways and windows may also be moved and their sizes changed; rooms, doorways and windows may be added and removed at will. At all times, the rooms on a plan should satisfy two constraints:

1. There are no 'island' rooms: all rooms should be connected through walls.
2. There is always a path between any two rooms - a virtual character can get from any room to any room.

Task 4

(a) Extend the design view to allow object selection.

  i) The user should be able to select a room, doorway or window. This could be a through a list or interactively by clicking on the visual representation of the object.

(b) Extend the design view to allow editing of existing rooms.

  i) Room sizes can be changed; ensure a minimum room size and that enlarging a room does not create an overlap with another room.

  ii) Room positions can be modified; ensure rooms do not become islands or end up overlapping other rooms.

  iii) Allow the user to change the materials for floor, walls and ceiling of a room.

(c) Extend the design view to allow editing of existing doorways and windows.

  i) Doorways sizes can be changed; enforce minimum and maximum doorway sizes such that a doorway is not larger than the wall it sits on. The same applies to windows.

  ii) Doorway and window can be moved along walls.

[20 marks]

Object Placement

While in design view, **BiniTool** should provide an **object placement** mode, where objects such as furniture, doors, lighting fixtures and decorations can be added to the floor plan, to embellish the design. The use will be able to select objects from a catalogue and place them in the scene. Objects may be rotated, scaled and moved.

Catalogue objects are labelled either **ground** or **float**. These labels help in the placement of objects from the overhead view, where depth perception is elusive due to the orthographic nature of the projection. Objects labelled ground come to rest on the floor unless another object also occupies the same space. Ground-tagged objects also possess another property - the vertical stacking priority. If two objects possess the same stacking priority, they repel each other horizontally, in the x-z plane. If their priorities are different, the higher priority object sits at the bottom of the stack, while the object with lower priority sits on top. A bounding box should be created for every catalogued object, either automatically or manually, and used to detect overlap.

Float-tagged objects are affixed to walls or ceilings (e.g. lighting fixtures, top-row cupboards and shelves). Thus, they do not come to rest on the ground but rather remain floating in mid-air.

Task 5

(a) Add placement mode to the **BiniTool** design view

  i) Add a catalogue of objects from which the user can select objects to place in the scene.

  ii) Placed objects may be scaled, translated and rotated.

(b) Tag catalogue objects with additional properties.

  i) Objects possess a bounding box that is used for collision detection, for the purpose of avoiding overlap.

  ii) Objects are labelled ground or float. Ground-tagged objects come to rest on the floor; float-tagged object float in mid-air.

  iii) Objects are given vertical-stacking priorities.

(c) Implement automatic overlap detection and removal.

  i) Objects with overlapping bounding boxes constitute overlap.

  ii) Overlapping ground-tagged objects repel each other; the direction is determined by their respective vertical stacking priorities.

[15 marks]

Modular Objects and Grouping

The tool should provide the ability to group objects; changes to a group object are extended to the entire group. For instance, changing the surface material of a group object (e.g. cupboard from pine to oak), should change that of the entire group. This also applies to operations such as scaling, translation and rotation.

Furthermore, modular objects (e.g. cupboards and shelves) should be extended with an orientation property - a vector that specifies the correct direction the object should face when placed against a wall. Placing a modular object in the scene should automatically make it snap to the closest wall and other modular objects, and look in the correct direction.

Task 6

(a) Extend design view to support object grouping.

- i) Operations on a group are applied to all its members.
- ii) Objects can be added or removed from a group at will.
- iii) Objects removed from a group retain their current properties; however, further changes to the group will not affect them.

(b) Add support for modular objects.

- i) Annotate modular objects with an orientation vector.
- ii) Placing modular objects will automatically make them snap to the closest wall.
- iii) Modular objects will also snap to other modular objects.

[5 marks]

Guides and Helpers

In design view, the tool should provide a number of visual aids such as dimension lines, labels and grid lines. Visual cues should always be aligned with the x-z plane and overlap all other visual elements, with the exception of gridlines. The user should be able to set the resolution of the grid to match real-world units, e.g. $10cm$, $1m$ and so on.

Grid lines act as a natural harness for snapping operations, which the user can toggle on and off. When snapping is enabled, translation operations on an object are automatically rounded to the grid resolution, and the object's anchor point (the origin in the object's local coordinate system) is aligned with the grid lines. For instance, if the grid resolution is $10cm$ then objects are translated in steps of $10cm$ and the object's anchor point is a point on the grid lines. Moreover, when snap is enabled, object rotations switch from a per-degree accuracy to right-angle rotations.

Task 7

(a) Add gridlines to the design view.

  i) Draw a wireframe square lattice (or grid graph) aligned with the floor plane.

  ii) The spacing of vertices is user-specified; the user should enter this property in metric units such as centimetres or metres.

  iii) The user can switch grid visualisation on and off.

(b) Add dimension lines to rooms in design view.

  i) Dimension lines should appear on one of every pair of opposite walls.

  ii) Dimension lines should appear on doorways and windows.

  iii) User can switch dimension lines on and off.

(c) Add snapping to object manipulation operations.

  i) Translation operations snap to grid.

  ii) Rotation operations snap to $90°$ turns.

  iii) User can turn snapping on and off.

[10 marks]

Exploration View

The second mode supported by **BiniTool** is **exploration view**, where users are allowed to walk through and explore their creations. The tool should support two navigation modalities: first-person walkthrough and free camera. For first-person mode, the user should be allowed to input the height of their avatar. The camera is then placed at eye level and a *WASD* + *mouse* control scheme used to move around the scene. In this mode, proper collision detection should be implemented to stop a user from exiting rooms through walls. Furthermore, for doorways with actual doors, the user should have to open the door before passage is allowed. In free-roam mode, the camera is not limited to the avatar height and not constrained by the actual walls. However, the tool should let the user define the scene bounds (as a slab) and constrain the camera to them.

Before users can take to exploring the scene, they can tweak the current time of day, which determines the position of the sun (or the moon) and the skybox selection. The day should be divided into four periods, each with its own skybox and light source placement: sunrise/morning, day, sunset/evening and night. There is no need for the time-of-day system to be dynamic; both skybox and light sources are fixed once exploration mode kicks off.

Task 8

(a) Add exploration view to your implementation.

i) Add first-person walkthrough navigation with collision detection and user-specified camera height.

ii) Add free-roam mode with user-specified scene boundaries.

(b) Add time-of-day system to exploration mode

i) Use different skyboxes for the four time periods making up the day.

ii) Add light sources to simulate the sun and the moon, depending on the day time.

[20 marks]

Final Notes

Feel free to experiment and try to enjoy this assignment! Discuss ideas or difficulties with your classmates, but do not leave the assignment for the last week. You are allowed to tweak the mechanics of the system; how you design the user interface and interactions is largely up to you. The effort put into this assignment will be duly noted during marking. **Good luck!**