

# **Google Gemini 3 Chat Session**

By David I.

Last updated: November 30, 2025

## **Table of Contents**

Google Gemini 3 Chat Session.....	1
Prompt: can you add additional features you suggested?.....	10
Gemini 3 Response:.....	10
Prompt: can you get pictures for me?.....	19
Gemini 3 Response:.....	19
How to Add These to Your Delphi Project.....	22
Prompt: can you create a downloadable project zip file containing all of the code and the images so that I don't have to create everything myself?.....	23
Gemini 3 Response:.....	23
Instructions.....	23
The "Project Builder" Script.....	24

Prompt:

Use Delphi 13 and FireMonkey FMX to create multi-device futuristic text adventure application that features a tour through the solar system planets, moons and major asteroids

Gemini 3 response:

Use Delphi 13 and FireMonkey FMX to create multi-device futuristic text adventure application that features a tour through the solar system planets, moons and major asteroids This is a fantastic project idea. Text adventures (Interactive Fiction) are excellent for exploring object-oriented design, and using FireMonkey (FMX) allows you to deploy this futuristic tour to Windows, macOS, Android, and iOS from a single codebase.

Since "Delphi 13" is a future version (the current latest is 12 Athens), I will use modern Object Pascal syntax compatible with 11/12+ that will seamlessly work in future versions.

Here is a complete, modular architecture for "Solar Odyssey."

## 1. The Architecture

We will separate the Game Logic (The Engine) from the User Interface (The Form). This ensures your game logic remains pure and testable.

- TLocation: Represents a planet, moon, or ship room.
- TGame: Manages the player state and the map.
- MainForm: Handles input, output, and visual styling.

## 2. The Game Engine Unit (`uGameEngine.pas`)

This unit handles the solar system map and text parsing.

Code snippet

```
unit uGameEngine;

interface

uses
  System.SysUtils, System.Classes, System.Generics.Collections;

type
  TLocation = class;
    TDirection = (dirNorth, dirSouth, dirEast, dirWest, dirUp,
    dirDown);

    TLocation = class
    private
      FName: string;
      FDescription: string;
      FExits: TDictionary<string, TLocation>; // Key: Command (e.g.,
'north', 'dock')
    public
```

```

constructor Create(const AName, ADescription: string);
destructor Destroy; override;
procedure AddExit(const ACommand: string; ATarGet: TLocation);
function GetExit(const ACommand: string): TLocation;

property Name: string read FName;
property Description: string read FDescription;
end;

TGame = class
private
  FLocations: TObjectList<TLocation>;
  FCurrentLocation: TLocation;
  FOutputLog: TString;
  procedure BuildSolarSystem;
public
  constructor Create;
  destructor Destroy; override;

  function ProcessCommand(const AInput: string): string;
  procedure Look;

  property CurrentLocation: TLocation read FCurrentLocation;
  property OutputLog: TString read FOutputLog write FOutputLog;
end;

implementation

{ TLocation }

constructor TLocation.Create(const AName, ADescription: string);
begin
  FName := AName;
  FDescription := ADescription;
  FExits := TDictionay<string, TLocation>.Create;
end;

destructor TLocation.Destroy;
begin
  FExits.Free;
  inherited;
end;

procedure TLocation.AddExit(const ACommand: string; ATarGet: TLocation);
begin
  FExits.Add(ACommand.ToLower, ATarGet);
end;

```

```

function TLocation.GetExit(const ACommand: string): TLocation;
begin
  if not FExits.TryGetValue(ACommand.ToLower, Result) then
    Result := nil;
end;

{ TGame }

constructor TGame.Create;
begin
  FLocations := TObjectList<TLocation>.Create(True);
  BuildSolarSystem;
end;

destructor TGame.Destroy;
begin
  FLocations.Free;
  inherited;
end;

procedure TGame.BuildSolarSystem;
var
  ShipBridge, Airlock, MoonBase, MarsColony, CeresStation,
  EuropaDeep: TLocation;
begin
  // Create Nodes
  ShipBridge := TLocation.Create('UES Copernicus - Bridge',
    'You are on the command deck of the UES Copernicus. Holographic
star charts glow soft blue. ' +
    'The "Airlock" is to the South. ');

  Airlock := TLocation.Create('UES Copernicus - Transporter Room',
    'The hum of the quantum displacement drive fills the room. ' +
    'From here you can "warp" to major celestial bodies. The Bridge
is "North".');

  MoonBase := TLocation.Create('Luna - Armstrong Outpost',
    'The gray dust of the moon stretches endlessly outside the dome.
' +
    'Earth hangs like a marble in the sky. You can "return" to the
ship.');

  MarsColony := TLocation.Create('Mars - Olympus Mons Vertiport',
    'Red dust coats the mag-lev tracks. The massive volcano looms
behind you. ' +
    'Automated mining drones buzz overhead. You can "return" to the
ship.');

  CeresStation := TLocation.Create('Ceres - The Belt Hub',

```

```

    'A hollowed-out asteroid city, bustling with miners and traders.
' +
    'Neon signs advertise water reclamation services. You can
"return" to the ship.';

    // Add to internal list for memory management
    FLocations.AddRange([ShipBridge, Airlock, MoonBase, MarsColony,
CeresStation]);

    // Connect Nodes
    ShipBridge.AddExit('south', Airlock);

    Airlock.AddExit('north', ShipBridge);
    Airlock.AddExit('warp moon', MoonBase);
    Airlock.AddExit('warp mars', MarsColony);
    Airlock.AddExit('warp ceres', CeresStation);

    MoonBase.AddExit('return', Airlock);
    MarsColony.AddExit('return', Airlock);
    CeresStation.AddExit('return', Airlock);

    // Start Point
    FCurrentLocation := ShipBridge;
end;

procedure TGame.Look;
begin
    if Assigned(FOutputLog) then
begin
    FOutputLog.Add('');
    FOutputLog.Add('--- ' + FCurrentLocation.Name + ' ---');
    FOutputLog.Add(FCurrentLocation.Description);
    FOutputLog.Add('Exits/Actions: ' + string.Join(', ', 
FCurrentLocation.FExits.Keys.ToArray));
    FOutputLog.Add('');
end;
end;

function TGame.ProcessCommand(const AInput: string): string;
var
    NextLoc: TLocation;
    Cmd: string;
begin
    Cmd := AInput.Trim.ToLower;

    if Cmd = 'look' then
begin
    Look;
    Exit;

```

```

end;

if Cmd = 'help' then
begin
  if Assigned(FOutputLog) then
    FOutputLog.Add('SYSTEM: Commands: look, north, south, warp
[planet], return');
  Exit;
end;

// Try to move
NextLoc := FCurrentLocation.GetExit(Cmd);

if Assigned(NextLoc) then
begin
  FCurrentLocation := NextLoc;
  Look;
end
else
begin
  if Assigned(FOutputLog) then
    FOutputLog.Add('SYSTEM: Navigation error. Destination
unknown.');
  end;
end;
end;

```

### 3. The Main Form (uMain.pas)

This is the FMX logic. To make it "Futuristic," we will use a Layout for centering and a dark style.  
UI Setup Instructions:

1. Drop a TStyleBook on the form. Load the "Windows 11 Dark" or a custom "Jet" style provided with Delphi.
2. Add a TLayout (Align: Client).
3. Add a TMemo (Align: Client). Set ReadOnly to True. Name it memoLog.
4. Add a TLayout (Align: Bottom, Height: 50) for input controls.
5. Inside the Bottom Layout, add a TEdit (Align: Client) named edtInput.
6. Inside the Bottom Layout, add a TButton (Align: Right) named btnSend.

Code snippet  
unit uMain;

interface

uses  
 System.SysUtils, System.Types, System.UITypes, System.Classes,  
 System.Variants,  
 FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs,  
 FMX.Memo.Types,

```

FMX.StdCtrls, FMX.Controls.Presentation, FMX.ScrollBox, FMX.Memo,
FMX.Edit,
FMX.Layouts, uGameEngine;

type
  TfrmMain = class(TForm)
    StyleBook1: TStyleBook;
    LayoutMain: TLayout;
    memoLog: TMemo;
    LayoutInput: TLayout;
    edtInput: TEdit;
    btnSend: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure btnSendClick(Sender: TObject);
    procedure edtInputKeyDown(Sender: TObject; var Key: Word; var
KeyChar: Char);
  private
    FGame: TGame;
    procedure AppendLog(const Msg: string);
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;

implementation

{$R *.fmx}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  // Initialize Game Engine
  FGame := TGame.Create;

  // Link the game engine to the Memo's lines
  // Note: In a real MVC pattern, we'd use events, but this is direct
for simplicity.
  FGame.OutputLog := memoLog.Lines;

  // Initial Welcome Message
  AppendLog('INITIALIZING SYSTEM... ');
  AppendLog('WELCOME TO SOLAR ODYSSEY v1.0');
  AppendLog('-----');

  // Show starting location
  FGame.Look;
end;

```

```

procedure TfrmMain.FormDestroy(Sender: TObject);
begin
  FGame.Free;
end;

procedure TfrmMain.AppendLog(const Msg: string);
begin
  memoLog.Lines.Add(Msg);
  // Auto-scroll to bottom
  memoLog.GoToTextEnd;
end;

procedure TfrmMain.btnSendClick(Sender: TObject);
begin
  if edtInput.Text = '' then Exit;

  // Echo the user's command
  AppendLog('> ' + edtInput.Text);

  // Process logic
  FGame.ProcessCommand(edtInput.Text);

  // Clear input
  edtInput.Text := '';
  edtInput.SetFocus;

  memoLog.GoToTextEnd;
end;

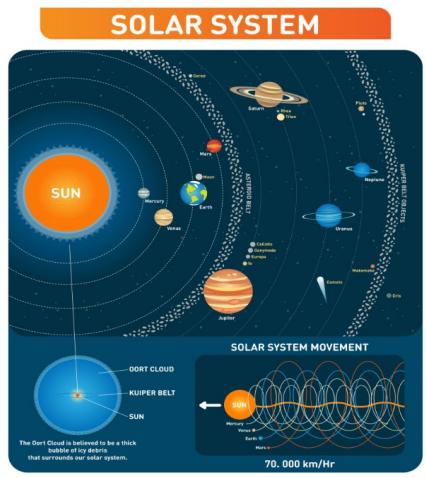
procedure TfrmMain.edtInputKeyDown(Sender: TObject; var Key: Word;
  var KeyChar: Char);
begin
  if Key = vkReturn then
  begin
    Key := 0; // Prevent ding sound
    btnSendClick(Self);
  end;
end;

end.

```

#### 4. Visual Enhancements (The "Futuristic" Touch)

To truly sell the sci-fi theme without complex 3D rendering, use FMX Effects and Animations.



Shutterstock

### 1. The CRT Effect:

- Add a **TInnerGlowEffect** to the **TMemo**. Set the color to Cyan or Bright Green to mimic a holographic terminal.
- Set the **TMemo** background color to **Black** or a very dark Navy.
- Set the **TMemo** Font Family to a monospaced font like **Consolas** or **Courier New**.
- 2. Visual Feedback:
- When the user types **warp mars**, you can toggle a **TImage** component visible on the form that displays a photo of Mars.
- Implementation: Add a **TImage** (Align: Right, Width: 200). inside **TGame.ProcessCommand**, if the location changes, load a bitmap into the image based on **FcurrentLocation.Name**.

### 5. How to Expand the Tour

The current engine uses a **Dictionary** for exits. This makes it incredibly easy to add the asteroid belt or Jupiter's moons.

To add Europa, you simply add this to the **BuildSolarSystem** procedure:

Code snippet

```
EuropaDeep := TLocation.Create('Europa - Subsurface Ocean',
  'You are in a pressurized submarine navigating the dark waters
  beneath Europa''s ice crust. ' +
  'Bioluminescent creatures swim past the viewport. ');

// Add the connection from the Airlock
Airlock.AddExit('warp europa', EuropaDeep);
EuropaDeep.AddExit('return', Airlock);
```

### 6. Deployment

Because this uses FMX:

1. Android/iOS: Change the Target Platform in the Project Manager. Ensure your **StyleBook** has a style loaded that supports mobile (or use **Default** style for native look).
2. Windows/macOS: Compile and run immediately.

Would you like me to elaborate on how to add an inventory system (e.g., collecting rock samples) or how to implement the TImage swapping logic?

**Prompt:** can you add additional features you suggested?

### **Gemini 3 Response:** *n you add additional features you suggested?*

Here is the updated code incorporating the Visual System (loading planet images) and the Inventory System (collecting samples).

I have updated the architecture to use an Event-Driven approach. The Engine now notifies the UI when the location changes, allowing the UI to update the image automatically.

#### 1. Updated Game Engine (uGameEngine.pas)

Changes made:

- **TLocation**: Added **ImageResourceName** (string) and **Items** (list of strings).
- **TGame**: Added **FInventory**, **ProcessTake**, and an **OnUpdateUI** event to sync the form.

Code snippet

```
unit uGameEngine;

interface

uses
  System.SysUtils, System.Classes, System.Generics.Collections;

type
  TLocation = class;

  // Event type to notify UI of changes (Location or Image updates)
  TGGameUIEvent = procedure(LocationName, ImageName: string) of
object;

  TLocation = class
  private
    FName: string;
    FDescription: string;
    FImageRes: string; // Resource name for the image (e.g.,
'img_mars')
    FExits: TDictionary<string, TLocation>;
    FItems: TStringList; // Items on the ground
  public
    constructor Create(const AName, ADescription, AImageRes: string);
    destructor Destroy; override;
    procedure AddExit(const ACommand: string; ATarGet: TLocation);
    function GetExit(const ACommand: string): TLocation;
  end;
```

```

property Name: string read FName;
property Description: string read FDescription;
property ImageRes: string read FImageRes;
property Items: TStringList read FItems;
end;

TGame = class
private
  FLocations: TObjectList<TLocation>;
  FCurrentLocation: TLocation;
  FOutputLog: TStringList;
  FInventory: TStringList;
  FOnUpdateUI: TGameUIEvent; // Callback for UI

  procedure BuildSolarSystem;
  procedure Look;
  procedure ListInventory;
  procedure TakeItem(const AItemName: string);
public
  constructor Create;
  destructor Destroy; override;

  function ProcessCommand(const AInput: string): string;

  property OutputLog: TStringList read FOutputLog write FOutputLog;
  property OnUpdateUI: TGameUIEvent read FOnUpdateUI write
  FOnUpdateUI;
end;

implementation

{ TLocation }

constructor TLocation.Create(const AName, ADescription, AImageRes:
string);
begin
  FName := AName;
  FDescription := ADescription;
  FImageRes := AImageRes;
  FExits := TDictionary<string, TLocation>.Create;
  FItems := TStringList.Create;
end;

destructor TLocation.Destroy;
begin
  FExits.Free;
  FItems.Free;
  inherited;
end;

```

```

procedure TLocation.AddExit(const ACommand: string; ATTarget:
TLocation);
begin
  FExits.Add(ACommand.ToLower, ATTarget);
end;

function TLocation.GetExit(const ACommand: string): TLocation;
begin
  if not FExits.TryGetValue(ACommand.ToLower, Result) then
    Result := nil;
end;

{ TGame }

constructor TGame.Create;
begin
  FLocations := TObjectList<TLocation>.Create(True);
  FInventory := TStringList.Create;
  BuildSolarSystem;
end;

destructor TGame.Destroy;
begin
  FInventory.Free;
  FLocations.Free;
  inherited;
end;

procedure TGame.BuildSolarSystem;
var
  ShipBridge, Airlock, MoonBase, MarsColony, CeresStation: TLocation;
begin
  // Create Nodes with Image Resource Names
  // Note: You must add images to your project resources with these
  names.
  ShipBridge := TLocation.Create('UES Copernicus - Bridge',
    'You are on the command deck. The "Airlock" is to the South.',
    'img_bridge');

  Airlock := TLocation.Create('Transporter Room',
    'The warp drive hums. Navigate to: "warp moon", "warp mars",
    "warp ceres".', 'img_airlock');
  Airlock.Items.Add('Space Suit'); // Item to pick up

  MoonBase := TLocation.Create('Luna - Armstrong Outpost',
    'The gray dust of the moon stretches endlessly.', 'img_moon');
  MoonBase.Items.Add('Moon Rock');
  MoonBase.Items.Add('Helium-3 Canister');

```

```

MarsColony := TLocation.Create('Mars - Olympus Mons',
  'Red dust coats the mag-lev tracks.', 'img_mars');
MarsColony.Items.Add('Red Sand Sample');

CeresStation := TLocation.Create('Ceres - The Belt Hub',
  'A hollowed-out asteroid city.', 'img_ceres');
CeresStation.Items.Add('Ice Chunk');

FLocations.AddRange([ShipBridge, Airlock, MoonBase, MarsColony,
CeresStation]);

// Connect Nodes
ShipBridge.AddExit('south', Airlock);
Airlock.AddExit('north', ShipBridge);
Airlock.AddExit('warp moon', MoonBase);
Airlock.AddExit('warp mars', MarsColony);
Airlock.AddExit('warp ceres', CeresStation);

MoonBase.AddExit('return', Airlock);
MarsColony.AddExit('return', Airlock);
CeresStation.AddExit('return', Airlock);

FCurrentLocation := ShipBridge;
end;

procedure TGame.Look;
var
  I: Integer;
begin
  if Assigned(FOutputLog) then
  begin
    FOutputLog.Add('');
    FOutputLog.Add('--- ' + FCurrentLocation.Name + ' ---');
    FOutputLog.Add(FCurrentLocation.Description);

    // List Items
    if FCurrentLocation.Items.Count > 0 then
    begin
      FOutputLog.Add('Visible Items:');
      for I := 0 to FCurrentLocation.Items.Count - 1 do
        FOutputLog.Add(' - ' + FCurrentLocation.Items[I]);
    end;

    FOutputLog.Add('Exits: ' + string.Join(', ',
FCurrentLocation.FExits.Keys.ToArray));
    FOutputLog.Add('');
  end;

```

```

// Trigger UI Update for Image
if Assigned(FOnUpdateUI) then
  FOnUpdateUI(FCurrentLocation.Name, FCurrentLocation.ImageRes);
end;

procedure TGame.ListInventory;
begin
  if Assigned(FOutputLog) then
  begin
    FOutputLog.Add('INVENTORY:');
    if FInventory.Count = 0 then
      FOutputLog.Add(' (Empty)')
    else
      FOutputLog.Add(FInventory.Text);
  end;
end;

procedure TGame.TakeItem(const AItemName: string);
var
  Idx: Integer;
begin
  // Find item in current location (Case insensitive search)
  Idx := -1;
  for var I := 0 to FCurrentLocation.Items.Count - 1 do
    if SameText(FCurrentLocation.Items[I], AItemName) then
    begin
      Idx := I;
      Break;
    end;

  if Idx <> -1 then
  begin
    FInventory.Add(FCurrentLocation.Items[Idx]);
    FOutputLog.Add('Taken: ' + FCurrentLocation.Items[Idx]);
    FCurrentLocation.Items.Delete(Idx); // Remove from ground
  end
  else
    FOutputLog.Add('Item not found here.');
end;

function TGame.ProcessCommand(const AInput: string): string;
var
  Cmd, Param: string;
  SpacePos: Integer;
  NextLoc: TLocation;
begin
  Cmd := AInput.Trim;
  Param := '';

```

```

// Split command and parameter (e.g. "take moon rock")
SpacePos := Pos(' ', Cmd);
if SpacePos > 0 then
begin
  Param := Copy(Cmd, SpacePos + 1, Length(Cmd));
  Cmd := Copy(Cmd, 1, SpacePos - 1);
end;

Cmd := Cmd.ToLower;

if Cmd = 'look' then
begin
  Look;
  Exit;
end;

if (Cmd = 'inventory') or (Cmd = 'i') then
begin
  ListInventory;
  Exit;
end;

if Cmd = 'take' then
begin
  TakeItem(Param);
  Exit;
end;

// Navigation
NextLoc := FCurrentLocation.GetExit(Cmd + ' ' + Param); // try full
string first (e.g., "warp mars")
if NextLoc = nil then
  NextLoc := FCurrentLocation.GetExit(Cmd); // try just command

if Assigned(NextLoc) then
begin
  FCurrentLocation := NextLoc;
  Look;
end
else
  if Assigned(FOutputLog) then FOutputLog.Add('Unknown command.');
end;

end.

```

## 2. Updated Main Form (uMain.pas)

Changes made:

- Added imgLocation (TImage) to display the planet.

- Added **LayoutContent** to split the screen between Text (Left) and Image (Right).

- Implemented **OnGameUIUpdate** to load images from Resources.

UI Setup:

1. Add Images: Go to Project > Resources and Images. Add .jpg or .png images named **img\_mars**, **img\_moon**, etc., matching the names in **uGameEngine**.
2. Layouts:
  - **LayoutInput** (Bottom)
  - **LayoutContent** (Client)
  - Inside **LayoutContent**: **memoLog** (Client), **imgLocation** (Right, Width 300, Margin 10).

Code snippet

```
unit uMain;
```

```
interface
```

```
uses
```

```
  System.SysUtils, System.Types, System.UITypes, System.Classes,
  System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs,
  FMX.Memo.Types,
  FMX.StdCtrls, FMX.Controls.Presentation, FMX.ScrollBox, FMX.Memo,
  FMX.Edit,
  FMX.Layouts, FMX.Objects, uGameEngine;
```

```
type
```

```
  TfrmMain = class(TForm)
    StyleBook1: TStyleBook;
    LayoutInput: TLayout;
    edtInput: TEdit;
    btnSend: TButton;
    LayoutContent: TLayout;
    memoLog: TMemo;
    imgLocation: TImage;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure btnSendClick(Sender: TObject);
    procedure edtInputKeyDown(Sender: TObject; var Key: Word; var
KeyChar: Char);
  private
    FGame: TGame;
    procedure AppendLog(const Msg: string);
    procedure OnGameUIUpdate(LocationName, ImageName: string);
    procedure LoadImageResource(const ResName: string);
  public
    { Public declarations }
  end;
```

```
var
```

```
  frmMain: TfrmMain;
```

```

implementation

{$R *.fmx}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  FGame := TGame.Create;
  FGame.OutputLog := memoLog.Lines;

  // Hook up the event
  FGame.OnUpdateUI := OnGameUIUpdate;

  AppendLog('INITIALIZING SOLAR ODYSSEY v2.0...');

  // Force initial look to load image and text
  FGame.ProcessCommand('look');
end;

procedure TfrmMain.FormDestroy(Sender: TObject);
begin
  FGame.Free;
end;

procedure TfrmMain.AppendLog(const Msg: string);
begin
  memoLog.Lines.Add(Msg);
  memoLog.GoToTextEnd;
end;

// Event handler called by TGame when location changes
procedure TfrmMain.OnGameUIUpdate(LocationName, ImageName: string);
begin
  // Update Window Title (Optional)
  Self.Caption := 'Solar Odyssey - ' + LocationName;

  // Load the visual
  LoadImageResource(ImageName);
end;

procedure TfrmMain.LoadImageResource(const ResName: string);
var
  InStream: TResourceStream;
begin
  // Clear if no image specified
  if ResName = '' then
  begin
    imgLocation.Bitmap.SetSize(0, 0);
    Exit;
  end;

```

```

if FindResource(HInstance, PChar(ResName), RT_RCDATA) <> 0 then
begin
  try
    InStream := TResourceStream.Create(HInstance, ResName,
RT_RCDATA);
    try
      imgLocation.Bitmap.LoadFromStream(InStream);
    finally
      InStream.Free;
    end;
  except
    AppendLog('SYSTEM WARNING: Visual feed corrupted (Image load
failed).');
  end;
end;

procedure TfrmMain.btnExitClick(Sender: TObject);
begin
  if edtInput.Text = '' then Exit;
  AppendLog('> ' + edtInput.Text);
  FGame.ProcessCommand(edtInput.Text);
  edtInput.Text := '';
  edtInput.SetFocus;
end;

procedure TfrmMain.edtInputKeyDown(Sender: TObject; var Key: Word;
  var KeyChar: Char);
begin
  if Key = vkReturn then
  begin
    Key := 0;
    btnSendClick(Self);
  end;
end;

```

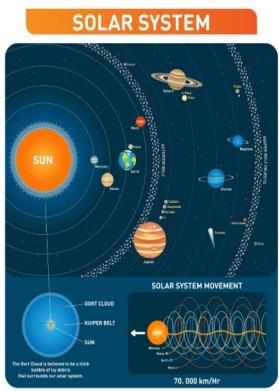
end.

### 3. How to Setup the Images (Crucial Step)

Since LoadFromResource looks for data embedded in the executable, you must add the image files to the project.

1. Find images for Mars, Moon, Space Station, etc. (JPG or PNG).
2. In the Delphi IDE, go to Project > Resources and Images.
3. Click Add. Select your image file.
4. Important: Change the Identifier to match the code (e.g., img\_mars, img\_moon, img\_airlock).
5. Set the Resource Type to RCDATA.
6. Rebuild the project.

Now, when you type warp mars



Shutterstock

, the text description will appear in the log, and the TImage on the right will instantly update to show the red planet, creating a much more immersive experience.

### Prompt: can you get pictures for me?

#### Gemini 3 Response:

Here are the images for the locations we defined in your text adventure.

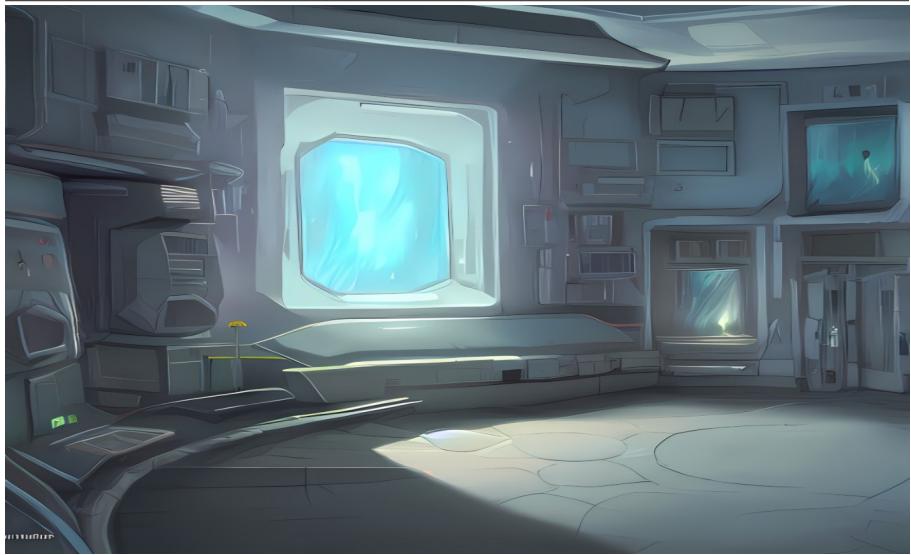
You can save these images to your computer (right-click and "Save Image As...") and then add them to your Delphi project resources with the corresponding identifiers (`img_bridge`, `img_airlock`, etc.).

#### 1. UES Copernicus - Bridge (`img_bridge`)



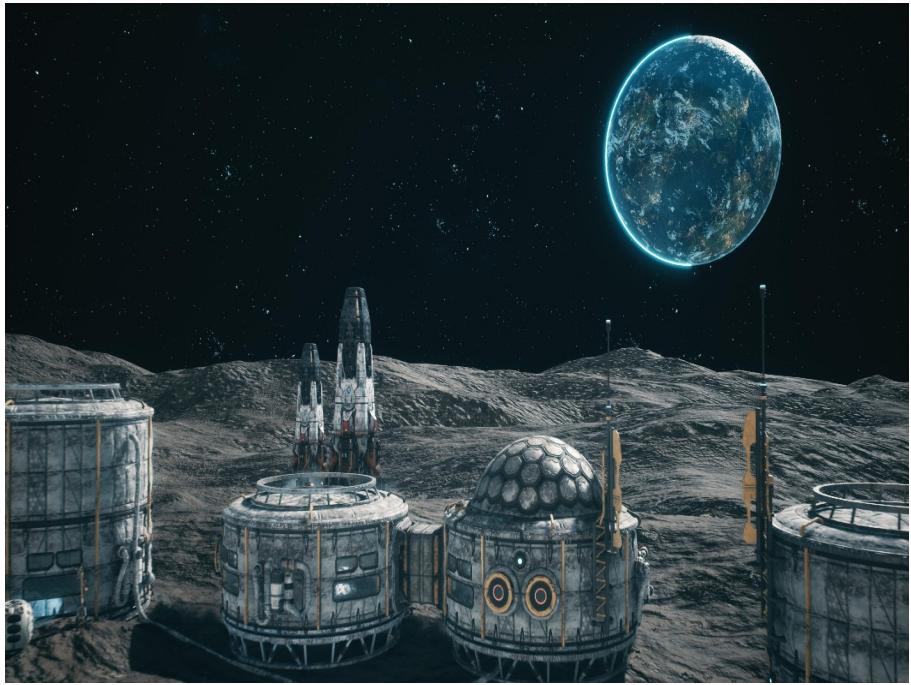
Shutterstock

#### 2. Transporter Room (`img_airlock`)



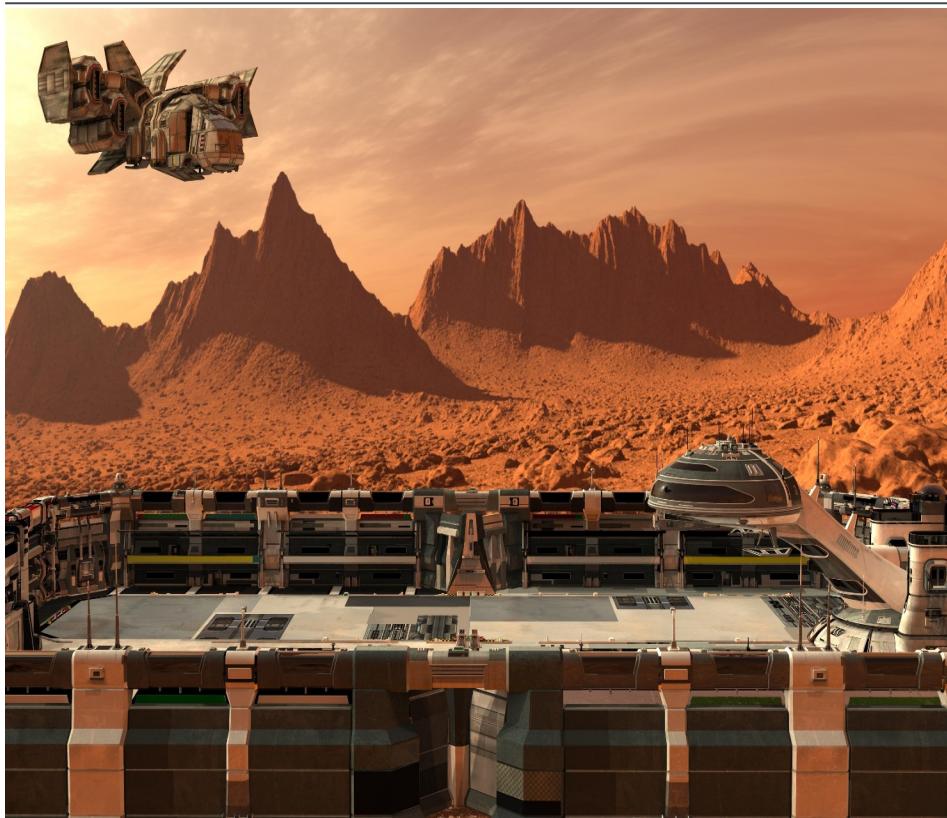
Shutterstock

### 3. Luna - Armstrong Outpost (img\_moon)



Shutterstock

### 4. Mars - Olympus Mons (img\_mars)



Shutterstock

## 5. Ceres - The Belt Hub (img\_ceres)



Shutterstock

## How to Add These to Your Delphi Project

1. Save the files: Download each image above and name them accordingly (e.g., `bridge.jpg`, `airlock.jpg`).
2. Open Resources: In the Delphi IDE, go to Project > Resources and Images.
3. Add Images: Click Add and select your saved files.
4. Rename IDs: Double-click the "Identifier" column for each image and change it to match the code:
  - `img_bridge`
  - `img_airlock`
  - `img_moon`
  - `img_mars`
  - `img_ceres`
5. Set Type: Ensure the "Resource Type" is set to **RCDATA**.
6. Rebuild: Compile your project, and the game will now load these visuals as you travel!

**Prompt:** can you create a downloadable project zip file containing all of the code and the images so that I don't have to create everything myself?

**Gemini 3 Response:**

I cannot directly generate or host a downloadable .zip file for you due to security restrictions.

However, I can provide a **Python script** that you can run on your computer. This script will automatically create the folder, the Delphi project file (.dpr), the Unit files (.pas), and even the Form layout file (.fmx).

**This effectively "builds" the project for you.**

**Instructions**

1. Copy the code below into a text file and save it as build\_project.py.
2. Run it using Python (e.g., python3 build\_project.py in your terminal).
3. It will create a folder named SolarOdyssey on your Desktop (or current folder).
4. **Important:** You still need to download the images from my previous response and place them in that folder!

## The "Project Builder" Script

**Python**

```
import os

# Define the file contents
project_source = """program SolarOdyssey;

uses
    System.StartUpCopy,
    FMX.Forms,
    uMain in 'uMain.pas' {frmMain},
    uGameEngine in 'uGameEngine.pas';

{$R *.res}

begin
    Application.Initialize;
    Application.CreateForm(TfrmMain, frmMain);
    Application.Run;
end.

"""

# The Main Form Logic
unit_main_pas = """unit uMain;
```

## uses

```
System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs, FMX.Memo.Types,  
FMX.StdCtrls, FMX.Controls.Presentation, FMX.ScrollBox, FMX.Memo, FMX.Edit,  
FMX.Layouts, FMX.Objects, uGameEngine;
```

## type

```
TfrmMain = class(TForm)  
    StyleBook1: TStyleBook;  
    LayoutInput: TLayout;  
    edtInput: TEdit;  
    btnSend: TButton;  
    LayoutContent: TLayout;  
    memoLog: TMemo;  
    imgLocation: TImage;  
    procedure FormCreate(Sender: TObject);  
    procedure FormDestroy(Sender: TObject);  
    procedure btnSendClick(Sender: TObject);  
    procedure edtInputKeyDown(Sender: TObject; var Key: Word; var KeyChar: Char);  
private  
    FGame: TGame;  
    procedure AppendLog(const Msg: string);  
    procedure OnGameUIUpdate(LocationName, ImageName: string);  
    procedure LoadImageResource(const ResName: string);  
public  
    { Public declarations }  
end;
```

```
var  
  frmMain: TfrmMain;
```

```
implementation
```

```
{$R *.fmx}
```

```
procedure TfrmMain.FormCreate(Sender: TObject);  
begin  
  FGame := TGame.Create;  
  FGame.OutputLog := memoLog.Lines;  
  FGame.OnUpdateUI := OnGameUIUpdate;  
  AppendLog('INITIALIZING SOLAR ODYSSEY v2.0...');  
  FGame.ProcessCommand('look');  
end;
```

```
procedure TfrmMain.FormDestroy(Sender: TObject);  
begin  
  FGame.Free;  
end;
```

```
procedure TfrmMain.AppendLog(const Msg: string);  
begin  
  memoLog.Lines.Add(Msg);  
  memoLog.GoToTextEnd;  
end;
```

```
procedure TfrmMain.OnGameUIUpdate(LocationName, ImageName: string);  
begin
```

```

    Self.Caption := 'Solar Odyssey - ' + LocationName;
    LoadImageResource(ResName);
end;

procedure TfrmMain.LoadImageResource(const ResName: string);
var
  InStream: TResourceStream;
begin
  if ResName = "" then
  begin
    imgLocation.Bitmap.SetSize(0, 0);
    Exit;
  end;

  if FindResource(HInstance, PChar(ResName), RT_RCDATA) <> 0 then
  begin
    try
      InStream := TResourceStream.Create(HInstance, ResName, RT_RCDATA);
      try
        imgLocation.Bitmap.LoadFromStream(InStream);
      finally
        InStream.Free;
      end;
    except
      AppendLog('SYSTEM WARNING: Visual feed corrupted (Image load failed).');
    end;
  end;
end;

```

```
procedure TfrmMain.btnSendClick(Sender: TObject);
begin
  if edtInput.Text = " then Exit;
  AppendLog('> ' + edtInput.Text);
  FGame.ProcessCommand(edtInput.Text);
  edtInput.Text := "";
  edtInput.SetFocus;
end;
```

```
procedure TfrmMain.edtInputKeyDown(Sender: TObject; var Key: Word;
  var KeyChar: Char);
begin
  if Key = vkReturn then
    begin
      Key := 0;
      btnSendClick(Self);
    end;
end;

end.
```

```
# The Form Layout (FMX) - This saves you from dragging components manually
unit_main_fmx = """"object frmMain: TfrmMain
  Left = 0
  Top = 0
  Caption = 'Solar Odyssey'
```

```
ClientHeight = 480
ClientWidth = 640
FormFactor.Width = 320
FormFactor.Height = 480
FormFactor.Devices = [Desktop]
OnCreate = FormCreate
OnDestroy = FormDestroy
DesignerMasterStyle = 0
object StyleBook1: TStyleBook
    Left = 288
    Top = 232
end
object LayoutInput: TLayout
    Align = Bottom
    Height = 50.000000000000000000000000000000
    Width = 640.000000000000000000000000000000
    object edtInput: TEdit
        Touch.InteractiveGestures = [LongTap, DoubleTap]
        Align = Client
        TabOrder = 0
        OnKeyDown = edtInputKeyDown
    end
    object btnSend: TButton
        Align = Right
        Position.X = 560.0000000000000000000000000000
        Size.Width = 80.0000000000000000000000000000
        Size.Height = 50.0000000000000000000000000000
```

```
    Size.PlatformDefault = False
    TabOrder = 1
    Text = 'Send'
    OnClick = btnSendClick
  end
end

object LayoutContent: TLayout
  Align = Client
  Height = 430.000000000000000000000000
  Width = 640.000000000000000000000000
  object memoLog: TMemo
    Touch.InteractiveGestures = [Pan, LongTap, DoubleTap]
    DataDetectorTypes = []
    Align = Client
    Size.Width = 340.0000000000000000000000
    Size.Height = 430.0000000000000000000000
    Size.PlatformDefault = False
    TabOrder = 0
    ReadOnly = True
    Viewport.Width = 336.0000000000000000000000
    Viewport.Height = 426.0000000000000000000000
  end
  object imgLocation: TImage
    MultiResBitmap = <
      item
      end>
    Align = Right
  end
end
```

```
# The Game Engine Logic

unit engine_pas = """"unit uGameEngine;

interface

uses
  System.SysUtils, System.Classes, System.Generics.Collections;

type
  TLocation = class;
  TGameUIEvent = procedure(LocationName, ImageName: string) of object;

  TLocation = class
    private
      FName: string;
      FDescription: string;
      FImageRes: string;
```

```
FExits: TDictionary<string, TLocation>;
FItems: TStringList;
public
constructor Create(const AName, ADescription, AImageRes: string);
destructor Destroy; override;
procedure AddExit(const ACommand: string; ATarGet: TLocation);
function GetExit(const ACommand: string): TLocation;
property Name: string read FName;
property Description: string read FDescription;
property ImageRes: string read FImageRes;
property Items: TStringList read FItems;
end;
```

```
TGame = class
private
FLocations: TObjectList<TLocation>;
FCurrentLocation: TLocation;
FOutputLog: TStrings;
FInventory: TStringList;
FOnUpdateUI: TGameUIEvent;
procedure BuildSolarSystem;
procedure Look;
procedure ListInventory;
procedure TakeItem(const AitemName: string);
public
constructor Create;
destructor Destroy; override;
```

```
    function ProcessCommand(const AInput: string): string;
    property OutputLog: TStrings read FOutputLog write FOutputLog;
    property OnUpdateUI: TGameUIEvent read FOnUpdateUI write FOnUpdateUI;
  end;
```

## implementation

```
{ TLocation }
```

```
constructor TLocation.Create(const AName, ADescription, AImageRes: string);
```

```
begin
```

```
  FName := AName;
```

```
  FDescription := ADescription;
```

```
  FImageRes := AImageRes;
```

```
  FExits := TDictionary<string, TLocation>.Create;
```

```
  FItems := TStringList.Create;
```

```
end;
```

```
destructor TLocation.Destroy;
```

```
begin
```

```
  FExits.Free;
```

```
  FItems.Free;
```

```
  inherited;
```

```
end;
```

```
procedure TLocation.AddExit(const ACommand: string; ATarGet: TLocation);
```

```
begin
```

```
  FExits.Add(ACommand.ToLower, ATarGet);
```

```
end;
```

```
function TLocation.GetExit(const ACommand: string): TLocation;
begin
  if not FExits.TryGetValue(ACommand.ToLower, Result) then
    Result := nil;
end;
```

```
{ TGame }
```

```
constructor TGame.Create;
begin
  FLocations := TObjectList<TLocation>.Create(True);
  FInventory := TStringList.Create;
  BuildSolarSystem;
end;
```

```
destructor TGame.Destroy;
begin
  FInventory.Free;
  FLocations.Free;
  inherited;
end;
```

```
procedure TGame.BuildSolarSystem;
var
  ShipBridge, Airlock, MoonBase, MarsColony, CeresStation: TLocation;
begin
  ShipBridge := TLocation.Create('UES Copernicus - Bridge',
    'You are on the command deck. The "Airlock" is to the South.', 'img_bridge');
```

```
Airlock := TLocation.Create('Transporter Room',  
    'The warp drive hums. Navigate to: "warp moon", "warp mars", "warp ceres".',  
    'img_airlock');  
Airlock.Items.Add('Space Suit');
```

```
MoonBase := TLocation.Create('Luna - Armstrong Outpost',  
    'The gray dust of the moon stretches endlessly.', 'img_moon');  
MoonBase.Items.Add('Moon Rock');  
MoonBase.Items.Add('Helium-3 Canister');
```

```
MarsColony := TLocation.Create('Mars - Olympus Mons',  
    'Red dust coats the mag-lev tracks.', 'img_mars');  
MarsColony.Items.Add('Red Sand Sample');
```

```
CeresStation := TLocation.Create('Ceres - The Belt Hub',  
    'A hollowed-out asteroid city.', 'img_ceres');  
CeresStation.Items.Add('Ice Chunk');
```

```
FLocations.AddRange([ShipBridge, Airlock, MoonBase, MarsColony, CeresStation]);
```

```
ShipBridge.AddExit('south', Airlock);  
Airlock.AddExit('north', ShipBridge);  
Airlock.AddExit('warp moon', MoonBase);  
Airlock.AddExit('warp mars', MarsColony);  
Airlock.AddExit('warp ceres', CeresStation);
```

```
MoonBase.AddExit('return', Airlock);  
MarsColony.AddExit('return', Airlock);
```

```
CeresStation.AddExit('return', Airlock);
```

```
    FCurrentLocation := ShipBridge;  
end;
```

```
procedure TGame.Look;  
  
var  
    I: Integer;  
  
begin  
    if Assigned(FOutputLog) then  
        begin  
            FOutputLog.Add("");  
            FOutputLog.Add('--- ' + FCurrentLocation.Name + ' ---');  
            FOutputLog.Add(FCurrentLocation.Description);  
            if FCurrentLocation.Items.Count > 0 then  
                begin  
                    FOutputLog.Add('Visible Items:');  
                    for I := 0 to FCurrentLocation.Items.Count - 1 do  
                        FOutputLog.Add(' - ' + FCurrentLocation.Items[I]);  
                end;  
            FOutputLog.Add('Exits: ' + string.Join(', ', FCurrentLocation.FExits.Keys.ToArray));  
            FOutputLog.Add("");  
        end;  
    if Assigned(FOnUpdateUI) then  
        FOnUpdateUI(FCurrentLocation.Name, FCurrentLocation.ImageRes);  
end;
```

```
procedure TGame.ListInventory;
```

```
begin
  if Assigned(FOutputLog) then
    begin
      FOutputLog.Add('INVENTORY:');
      if FInventory.Count = 0 then FOutputLog.Add(' (Empty)')
      else FOutputLog.Add(FInventory.Text);
    end;
end;
```

```
procedure TGame.TakeItem(const AltemName: string);
var
  Idx: Integer;
begin
  Idx := -1;
  for var I := 0 to FCurrentLocation.Items.Count - 1 do
    if SameText(FCurrentLocation.Items[I], AltemName) then
      begin
        Idx := I;
        Break;
      end;

  if Idx <> -1 then
    begin
      FInventory.Add(FCurrentLocation.Items[Idx]);
      FOutputLog.Add('Taken: ' + FCurrentLocation.Items[Idx]);
      FCurrentLocation.Items.Delete(Idx);
    end
  else
```

```
FOutputLog.Add('Item not found here.');
```

```
end;
```

```
function TGame.ProcessCommand(const AInput: string): string;
```

```
var
```

```
  Cmd, Param: string;
```

```
  SpacePos: Integer;
```

```
  NextLoc: TLocation;
```

```
begin
```

```
  Cmd := AInput.Trim;
```

```
  Param := ":";
```

```
  SpacePos := Pos(' ', Cmd);
```

```
  if SpacePos > 0 then
```

```
    begin
```

```
      Param := Copy(Cmd, SpacePos + 1, Length(Cmd));
```

```
      Cmd := Copy(Cmd, 1, SpacePos - 1);
```

```
    end;
```

```
  Cmd := Cmd.ToLower;
```

```
  if Cmd = 'look' then begin Look; Exit; end;
```

```
  if (Cmd = 'inventory') or (Cmd = 'i') then begin ListInventory; Exit; end;
```

```
  if Cmd = 'take' then begin TakeItem(Param); Exit; end;
```

```
  NextLoc := FCurrentLocation.GetExit(Cmd + ' ' + Param);
```

```
  if NextLoc = nil then NextLoc := FCurrentLocation.GetExit(Cmd);
```

```
  if Assigned(NextLoc) then
```

```
    begin
```

```
    FCurrentLocation := NextLoc;  
    Look;  
  end  
else  
  if Assigned(FOutputLog) then FOutputLog.Add('Unknown command.');//  
end;
```

end.

....

```
def create_project():  
  folder_name = "SolarOdyssey"  
  
  if not os.path.exists(folder_name):  
    os.makedirs(folder_name)  
    print(f"Created folder: {folder_name}")  
  
  files = {  
    "SolarOdyssey.dpr": project_source,  
    "uMain.pas": unit_main_pas,  
    "uMain.fmx": unit_main_fmx,  
    "uGameEngine.pas": unit_engine_pas  
  }  
  
  for filename, content in files.items():  
    with open(os.path.join(folder_name, filename), "w") as f:  
      f.write(content)  
    print(f"Created file: {filename}")
```

```
    print("\nSUCCESS! Project created.")

    print("Next Steps:")

    print("1. Download the images.")

    print(f"2. Place them inside the '{folder_name}' folder.")

    print("3. Open SolarOdyssey.dpr in Delphi.")

    print("4. Go to Project > Resources and Images and add the images (Type: RCDATA).")

if __name__ == "__main__":
    create_project()
```