LAND OF DESIRE

Introduction

Land of Desire is a sex simulator similar to other titles in the genre such as *3D Sex Villa* or *Honey Select*, with the particularity of being free, open-source, and built on a game engine that shares these same qualities. The goal is to make the game easy to modify and, hopefully, allow it to survive over time if a community forms around it. It is also intended to serve as a foundation for those who want to create their own games in the same genre.

Important

If you plan to modify the game — whether it's adding clothes, animations, or anything else — it's strongly recommended to use the same version of the engine to avoid potential compatibility issues.

Technologies Used

- **Engine**: UPBGE 0.36.1
- **Modeling**: The game engine itself, which is a modified version of Blender
- **Icons**: All game icons were created using *LibreOffice Draw*, version 24.2.7.2 from the LibreOffice suite

---

Folder Structure

```
Land of desire/
├── Characters/
│   ├── Female/
│   └── Male/
│       ├── Clothes/
│       │   ├── Pants/
│       │   ├── Shoes/
│       │   ├── Textures/
│       │   └── Torso/
│       ├── Exported/
│       ├── Eyebrowns/
│       ├── Eyewear/
│       ├── Hairs/
│       ├── Piercings/
│       └── Skin_Tectures/
├── Data/
├── Icons/
├── Python/
├── Scenes/
├── icons.odg
└── all_game.blend
```

Programming

The game includes Python scripts and uses both UPBGE's **Logic Bricks** and **Logic Nodes** systems working together.

Character Editor System

When selecting the type of character to create (male or female), the corresponding `.blend` file — either `Female_char_Editor.blend` or `Male_char_Editor.blend` — is loaded from the `Characters` folder.
These files, upon loading, automatically import the contents of `Female_char_template.blend` or `Male_char_template.blend`, depending on the selected gender. The system then reads the shape keys of each mesh and the contents of the clothing folders to dynamically generate the visible user interface in the editor.

Body Modification (Shape Keys)

The base character has multiple shape keys used to alter body shape (such as thickness, height, etc.). Each slider in the menu adjusts the value of a specific shape key through the Python script `shape_key_value_change.py`.
The name of the shape key to be modified is defined in the Empty object called `script_holder`, using the property `ShapeKeyToChange`.

Skin Color

Skin color is adjusted by modifying the hue, saturation, and brightness of a specific material’s texture. These parameters are handled by the script `change_skin_color.py`.

Clothing Load System

Each piece of clothing is stored in a separate `.blend` file containing a mesh fitted to the character.

Loading process:

- When a clothing item is selected, the file is loaded and the object is instantiated.
- It is automatically positioned on the character’s body.
- Clothing meshes must contain the same shape keys as the character’s body, already adjusted to match the current values.
  When the clothing mesh is imported, the shape key values from the character are automatically applied to it.
  This process is handled by the script `load_clothing_or_hair_or_eyebrows.py`, which is also used for loading hair.

Character Saving

The system uses the script `save_character.py` to save the following in real-time:

- Modified shape keys
- Skin color
- Selected clothing and hair

This allows changes to be preserved between sessions, provided a save system is integrated later.
All data is stored as a `.json` file inside the `Exported` folders located in either `Female` or `Male`, depending on the character being edited.

Main Scripts in the Character Editor

- `shape_key_value_change.py`: Modifies a specific shape key's value.
- `load_clothing_or_hair_or_eyebrows.py`: Loads and fits a clothing item.
- `change_skin_color.py`: Changes the character's skin texture.
- `save_character.py`: Saves the character as a `.json` file.

Scripts are triggered via Logic Bricks through button presses and scene loading (managed by the `script_holder` object).

Clothing

Clothing for both male and female characters is stored in their respective folders.

- The `Torso` folder is for upper body garments.
- The `Pants` folder is used for any item covering the legs or groin — including pants or underwear (only one can be worn at a time; for example, a character cannot wear both pants and boxers).
- The `Shoes` folder is for footwear such as sneakers, socks, or slippers.

Extensibility

Adding New Shape Keys

Create a new shape key in the base character file:
`Male_char_template.blend` or `Female_char_template.blend`, as appropriate.

Adding New Clothes

1. Model the clothing fitted to the base character.
2. Save the item as a `.blend` file (Blender/UPBGE format) in the correct folder.
3. Ensure the clothing has the same shape keys as the base character, with each key already adjusted to the mesh for its respective deformation state.
   (It is recommended to duplicate the base character and modify the mesh directly.)
4. Name the mesh according to its type:
    - `torso_up` for tops and bras
    - `pants` for pants, underwear, or thongs

    - `shoes` for footwear like slippers, flip-flops, etc.
5. Clothing meshes must be rigged to a copy of the appropriate character's armature, and saved in the correct folder for that character.
   This is essential for the game to locate the object both in the file system and within each `.blend` file.

Technical Notes

- Clothing does not use collision simulation. Instead, it is modeled with a small margin to avoid mesh clipping.
- The system checks if a clothing item is already loaded and removes duplicates from the scene.
- The HUD is made entirely from flat 3D objects.

Animations

If you want to create new animations, all you need to do is open the file `animations.blend` inside the `Characters` folder.
There, you'll find a set of flat and cube-shaped empties used as markers, along with a pair of characters (male and female) that use the same armatures as the character templates.
Simply animate the characters using any marker as a reference (e.g., sitting, leaning, etc.).
Both characters should be assigned the same animation name when applying the NLA (Nonlinear Animation).
Don't worry if Blender automatically adds `.001`, `.002`, and so on — the game is designed to handle that.

Animation names must follow the format `pose:area`, where `area` is the exact name of the marker where the pose is executed.
Examples:

- `bed/floor` for poses on a bed or the floor
- `table` for poses on a table
- `wall` for poses against a wall

Important:
Never modify the markers — do not change their shape, size, or name. They must remain consistent across all game files.

Environments / Scenes

To create a new scene, import the markers (empties) from `animations.blend` (ignore the character pair).
Use these empties as size references for the different elements in your scene.
Never change the size or shape of a marker. You are allowed to rotate and move them freely.
For example, if you model a bed and it ends up too large or too small compared to the `bed/floor` marker, modify the mesh of the bed — **not** the marker.

Once your scene is ready, save it as a `.blend` file in the `Scenes` folder along with a `.png` screenshot of the scene.
(This image is not yet used in this version but will eventually appear in the scene selection menu.)
That's all — scene selection buttons are generated automatically when the menu is loaded in the game.

Game File (`all_game.blend`)

The launcher's only job is to load `all_game.blend` in real time.
This file must already be set up with the main menu scene active and the camera correctly positioned.
(Remember this if you plan to modify anything.)

This file includes:

1. The main menu
2. The character and scene selection menu
3. The main gameplay scene

Configuration

Video settings are applied and saved using a Logic Nodes setup called `VideoSettings`, attached to the `ScriptHolder` object in the main menu.
Settings are applied through another Logic Nodes block named `ApplySettings`,

linked to the `ApplySettingsButton` in the same scene.
These settings are saved to a file called `variables.json` inside the `Data` folder.

Character and Scene Selection

This takes place in the `character_selector` scene (originally meant only for character selection but later adapted for both characters and scenes).

In this scene, you'll find two unused markers (one male and one female) that were initially intended to preview the selected characters — they are now obsolete and not used.
Character selection buttons are generated by the script `create_scene_buttons.py`.

Once both a male and a female character, as well as a scene, have been selected, the **Start Game** button becomes active.
Clicking it switches to the `game_scene` and applies its corresponding HUD (via an overlay scene).

Not every aspect of the game is covered in this documentation. I've been working on this project alone for about seven months, with help from ChatGPT, and my energy is not unlimited. Still, I hope this serves as a solid starting point if you want to build or expand upon it.